

# Direct3D 11 Graphics

Article • 01/26/2022

You can use Microsoft Direct3D 11 graphics to create 3-D graphics for games and scientific and desktop applications.

This section contains information about programming with Direct3D 11 graphics.

For more information, see [Direct3D 11 Features](#).

[ ] [Expand table](#)

<b>Supported</b>	
<b>Supported runtime environments</b>	Windows/C++
<b>Recommended programming languages</b>	C/C++
<b>Minimum supported client</b>	Windows 7
<b>Minimum supported server</b>	Windows Server 2008 R2

[ ] [Expand table](#)

<b>Topic</b>	<b>Description</b>
<a href="#">How to Use Direct3D 11</a>	This section demonstrates how to use the Direct3D 11 API to accomplish several common tasks.
<a href="#">What's new in Direct3D 11</a>	This section describes features added in Direct3D 11, Direct3D 11.1, and Direct3D 11.2.
<a href="#">Programming Guide for Direct3D 11</a>	The programming guide contains information about how to use the Direct3D 11 programmable pipeline to create realtime 3D graphics for games as well as scientific and desktop applications.
<a href="#">Direct3D 11 Reference</a>	This section contains the reference pages for Direct3D 11-based graphics programming.

In addition to Direct3D 11 being supported by Windows 7 and later and Windows Server 2008 R2 and later, Direct3D 11 is available down-level for Windows Vista with Service Pack 2 (SP2) and Windows Server 2008 by downloading [KB 971512](#).

For info about new Direct3D 11.1 features that are included with Windows 8, Windows Server 2012, and are partially available on Windows 7 and Windows Server 2008 R2 via the [Platform Update for Windows 7](#), see [Direct3D 11.1 Features](#) and the [Platform Update for Windows 7](#).

# How to Use Direct3D 11

Article • 08/19/2020

This section demonstrates how to use the Microsoft Direct3D 11 API to accomplish several common tasks.

Topic	Description
<a href="#">How To: Create a Reference Device</a>	This topic shows how to create a reference device that implements a highly accurate, software implementation of the runtime.
<a href="#">How To: Create a WARP Device</a>	This topic shows how to create a WARP device that implements a high speed software rasterizer.
<a href="#">How To: Create a Swap Chain</a>	This topic shows how to create a swap chain that encapsulates two or more buffers that are used for rendering and display.
<a href="#">How To: Enumerate Adapters</a>	This topic shows how to use Microsoft DirectX Graphics Infrastructure (DXGI) to enumerate the available graphics adapters on a computer.
<a href="#">How To: Get Adapter Display Modes</a>	This topic shows how to use DXGI to get the valid display modes associated with an adapter.
<a href="#">How To: Create a Device and Immediate Context</a>	This topic shows how to initialize a <a href="#">device</a> .
<a href="#">How To: Get the Device Feature Level</a>	This topic shows how to get the highest <a href="#">feature level</a> supported by a <a href="#">device</a> .
<a href="#">How to: Create a Vertex Buffer</a>	This topic shows how to initialize a static <a href="#">vertex buffer</a> , that is, a vertex buffer that does not change.
<a href="#">How to: Create an Index Buffer</a>	This topic shows how to initialize an <a href="#">index buffer</a> in preparation for rendering.
<a href="#">How to: Create a Constant Buffer</a>	This topic shows how to initialize a <a href="#">constant buffer</a> in preparation for rendering.
<a href="#">How to: Create a Texture</a>	This topic shows how to create a texture.
<a href="#">How to: Initialize a Texture Programmatically</a>	This topic has several examples showing how to initialize textures that are created with different types of usages.
<a href="#">How to: Initialize a Texture From a File</a>	This topic shows how to use Windows Imaging Component (WIC) to create the texture and the view separately.

Topic	Description
<a href="#">How to: Use dynamic resources</a>	You create and use dynamic resources when your app needs to change data in those resources. You can create textures and buffers for dynamic usage.
<a href="#">How To: Create a Compute Shader</a>	This topic shows how to create a compute shader.
<a href="#">How To: Design a Hull Shader</a>	This topics shows how to design a hull shader.
<a href="#">How To: Create a Hull Shader</a>	This topic shows how to create a hull shader.
<a href="#">How To: Initialize the Tessellator Stage</a>	This topic shows how to initialize the tessellator stage.
<a href="#">How To: Design a Domain Shader</a>	This topics shows how to design a domain shader.
<a href="#">How To: Create a Domain Shader</a>	This topic shows how to create a domain shader.
<a href="#">How To: Compile a Shader</a>	This topic shows how to use the <a href="#">D3DCompileFromFile</a> function at run time to compile shader code.
<a href="#">How to: Record a Command List</a>	This topic shows how to create and record a <a href="#">command list</a> .
<a href="#">How to: Play Back a Command List</a>	This topic shows how to play back a <a href="#">command list</a> .
<a href="#">How To: Check for Driver Support</a>	This topic shows how to determine whether multithreading features (including <a href="#">resource creation</a> and <a href="#">command lists</a> ) are supported for hardware acceleration.

## Related topics

[Direct3D 11 Graphics](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Create a Reference Device

Article • 08/23/2019

This topic shows how to create a reference device that implements a highly accurate, software implementation of the runtime. To create a reference device, simply specify that the device you are creating will use a reference driver. This example creates a device and a swap chain at the same time.

## To create a reference device

1. Define initial parameters for a swap chain.

```
DXGI_SWAP_CHAIN_DESC sd;
ZeroMemory( &sd, sizeof( sd ) );
sd.BufferCount = 1;
sd.BufferDesc.Width = 640;
sd.BufferDesc.Height = 480;
sd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
sd.BufferDesc.RefreshRate.Numerator = 60;
sd.BufferDesc.RefreshRate.Denominator = 1;
sd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
sd.OutputWindow = g_hWnd;
sd.SampleDesc.Count = 1;
sd.SampleDesc.Quality = 0;
sd.Windowed = TRUE;
```

2. Request a feature level that implements the features your application will need. A reference device can be successfully created for the Direct3D 11 runtime.

```
D3D_FEATURE_LEVEL FeatureLevels = D3D_FEATURE_LEVEL_11_0;
```

See more about feature levels in the [D3D\\_FEATURE\\_LEVEL](#) enumeration.

3. Create the device by calling [D3D11CreateDeviceAndSwapChain](#).

```
HRESULT hr = S_OK;
D3D_FEATURE_LEVEL FeatureLevel;

if( FAILED (hr = D3D11CreateDeviceAndSwapChain( NULL,
                                              D3D_DRIVER_TYPE_REFERENCE,
                                              NULL,
```

```
    0,
    &FeatureLevels,
    1,
    D3D11_SDK_VERSION,
    &sd,
    &g_pSwapChain,
    &g_pd3dDevice,
    &FeatureLevel,
    &g_pImmediateContext )))

{
    return hr;
}
```

You will need to supply the API call with the reference driver type from the [D3D\\_DRIVER\\_TYPE](#) enumeration. After the method succeeds, it will return a swap chain interface, a device interface, a pointer to the feature level that was granted by the driver, and an immediate context interface.

For information about limitations creating a reference device on certain feature levels, see [Limitations Creating WARP and Reference Devices](#).[How to Use Direct3D 11](#)

## Related topics

[Devices](#)

[How to Use Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Create a WARP Device

Article • 08/19/2020

This topic shows how to create a WARP device that implements a high speed software rasterizer. To create a WARP device, simply specify that the device you are creating will use a WARP driver. This example creates a device and a swap chain at the same time.

## To create a WARP device

1. Define initial parameters for a swap chain.

```
DXGI_SWAP_CHAIN_DESC sd;
ZeroMemory( &sd, sizeof( sd ) );
sd.BufferCount = 1;
sd.BufferDesc.Width = 640;
sd.BufferDesc.Height = 480;
sd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
sd.BufferDesc.RefreshRate.Numerator = 60;
sd.BufferDesc.RefreshRate.Denominator = 1;
sd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
sd.OutputWindow = g_hWnd;
sd.SampleDesc.Count = 1;
sd.SampleDesc.Quality = 0;
sd.Windowed = TRUE;
```

2. Request a feature level that implements the features your application will need. A WARP device can be successfully created for feature levels D3D\_FEATURE\_LEVEL\_9\_1 through D3D\_FEATURE\_LEVEL\_10\_1 and starting with Windows 8 for all feature levels.

```
D3D_FEATURE_LEVEL FeatureLevels = D3D_FEATURE_LEVEL_10_1;
```

See more about feature levels in the [D3D\\_FEATURE\\_LEVEL](#) enumeration.

3. Create the device by calling [D3D11CreateDeviceAndSwapChain](#).

```
HRESULT hr = S_OK;
if( FAILED (hr = D3D11CreateDeviceAndSwapChain( NULL,
                                                D3D_DRIVER_TYPE_WARP,
                                                NULL,
```

```
    0,
    &FeatureLevels,
    1,
    D3D11_SDK_VERSION,
    &sd,
    &g_pSwapChain,
    &g_pd3dDevice,
    &FeatureLevel,
    &g_pImmediateContext )))

{
    return hr;
}
```

You will need to supply the API call with the WARP driver type from the [D3D\\_DRIVER\\_TYPE](#) enumeration. After the method succeeds, it will return a swap chain interface, a device interface, a pointer to the feature level that was granted by the driver, and an immediate context interface.

For information about limitations creating a WARP device on certain feature levels, see [Limitations Creating WARP and Reference Devices](#).

## New for Windows 8

When a computer's primary display adapter is the "Microsoft Basic Display Adapter" (WARP adapter), that computer also has a second adapter. This second adapter is the render-only device that has no display outputs. For more info about the render-only device, see [new info in Windows 8 about enumerating adapters](#).

## Related topics

[Devices](#)

[How to Use Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Create a Swap Chain

Article • 08/19/2020

This topic shows how to create a swap chain that encapsulates two or more buffers that are used for rendering and display. They usually contain a front buffer that is presented to the display device and a back buffer that serves as the render target. After the immediate context is done rendering to the back buffer, the swap chain presents the back buffer by swapping the two buffers.

The swap chain defines several rendering characteristics including:

- The size of the render area.
- The display refresh rate.
- The display mode.
- The surface format.

Define the characteristics of the swap chain by filling in a [DXGI\\_SWAP\\_CHAIN\\_DESC](#) structure and initializing an [IDXGISwapChain](#) interface. Initialize a swap chain by calling [IDXGIFactory::CreateSwapChain](#) or [D3D11CreateDeviceAndSwapChain](#).

## Create a device and a swap chain

To initialize a device and swap chain, use one of the following two functions:

- Use the [D3D11CreateDeviceAndSwapChain](#) function when you want to initialize the swap chain at the same time as device initialization. This usually is the easiest option.
- Use the [D3D11CreateDevice](#) function when you have already created a swap chain using [IDXGIFactory::CreateSwapChain](#).

## Related topics

[Devices](#)

[How to Use Direct3D 11](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# How To: Enumerate Adapters

Article • 08/19/2020

This topic shows how to use Microsoft DirectX Graphics Infrastructure (DXGI) to enumerate the available graphics adapters on a computer. Direct3D 10 and 11 can use DXGI to enumerate adapters.

You generally need to enumerate adapters for these reasons:

- To determine how many graphics adapters are installed on a computer.
- To help you choose which adapter to use to create a Direct3D device.
- To retrieve an [IDXGIAAdapter](#) object that you can use to retrieve device capabilities.

## To enumerate adapters

1. Create an [IDXGIFactory](#) object by calling the [CreateDXGIFactory](#) function. The following code example demonstrates how to initialize an [IDXGIFactory](#) object.

```
IDXGIFactory * pFactory = NULL;  
  
CreateDXGIFactory(__uuidof(IDXGIFactory) ,(void**)&pFactory)
```

2. Enumerate through each adapter by calling the [IDXGIFactory::EnumAdapters](#) method. The *Adapter* parameter allows you to specify a zero-based index number of the adapter to enumerate. If no adapter is available at the specified index, the method returns [DXGI\\_ERROR\\_NOT\\_FOUND](#).

The following code example demonstrates how to enumerate through the adapters on a computer.

```
for (UINT i = 0;  
     pFactory->EnumAdapters(i, &pAdapter) != DXGI_ERROR_NOT_FOUND;  
     ++i)  
{ ... }
```

The following code example demonstrates how to enumerate all adapters on a computer.

### Note

For Direct3D 11.0 and later, we recommend that apps always use `IDXGIFactory1` and `CreateDXGIFactory1` instead.

C++

```
std::vector<IDXGIAdapter*> EnumerateAdapters(void)
{
    IDXGIAdapter * pAdapter;
    std::vector<IDXGIAdapter*> vAdapters;
    IDXGIFactory* pFactory = NULL;

    // Create a DXGIFactory object.
    if(FAILED(CreateDXGIFactory(__uuidof(IDXGIFactory) , (void**)&pFactory)))
    {
        return vAdapters;
    }

    for ( UINT i = 0;
          pFactory->EnumAdapters(i, &pAdapter) != DXGI_ERROR_NOT_FOUND;
          ++i )
    {
        vAdapters.push_back(pAdapter);
    }

    if(pFactory)
    {
        pFactory->Release();
    }

    return vAdapters;
}
```

## Related topics

[How to Use Direct3D 11](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# How To: Get Adapter Display Modes

Article • 08/19/2020

This topic shows how to use Microsoft DirectX Graphics Infrastructure (DXGI) to get the valid display modes associated with an adapter. DirectX 10 and 11 can use DXGI to get the valid display modes. Knowing the valid display modes ensures that your application can properly choose a valid full-screen mode.

## To get adapter display modes

1. Create an [IDXGIFactory](#) object and use it to enumerate the available adapters. For more information, see [How To: Enumerate Adapters](#).
2. Call [IDXGIAdapter::EnumOutputs](#) to enumerate the outputs for each adapter.

```
IDXGIOOutput* pOutput = NULL;  
HRESULT hr;  
  
hr = pAdapter->EnumOutputs(0,&pOutput);
```

3. Call [IDXGIOOutput::GetDisplayModeList](#) to retrieve an array of [DXGI\\_MODE\\_DESC](#) structures and the number of elements in the array. Each [DXGI\\_MODE\\_DESC](#) structure represents a valid display mode for the output.

```
UINT numModes = 0;  
DXGI_MODE_DESC* displayModes = NULL;  
DXGI_FORMAT format = DXGI_FORMAT_R32G32B32A32_FLOAT;  
  
// Get the number of elements  
hr = pOutput->GetDisplayModeList( format, 0, &numModes, NULL );  
  
displayModes = new DXGI_MODE_DESC[numModes];  
  
// Get the list  
hr = pOutput->GetDisplayModeList( format, 0, &numModes,  
displayModes );
```

## Related topics

[Devices](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Create a Device and Immediate Context

Article • 08/24/2021

This topic shows how to initialize a [device](#). Initializing a [device](#) is one of the first tasks that your application must complete before you can render your scene.

To create a device and immediate context

Fill out the [DXGI\\_SWAP\\_CHAIN\\_DESC](#) structure with information about buffer formats and dimensions. For more information, see [Creating a Swap Chain](#).

The following code example demonstrates how to fill in the [DXGI\\_SWAP\\_CHAIN\\_DESC](#) structure.

```
DXGI_SWAP_CHAIN_DESC sd;
ZeroMemory( &sd, sizeof( sd ) );
sd.BufferCount = 1;
sd.BufferDesc.Width = 640;
sd.BufferDesc.Height = 480;
sd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
sd.BufferDesc.RefreshRate.Numerator = 60;
sd.BufferDesc.RefreshRate.Denominator = 1;
sd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
sd.OutputWindow = g_hWnd;
sd.SampleDesc.Count = 1;
sd.SampleDesc.Quality = 0;
sd.Windowed = TRUE;
```

Using the `DXGI_SWAP_CHAIN_DESC` structure from step one, call `D3D11CreateDeviceAndSwapChain` to initialize the device and swap chain at the same time.

```
    numFeatureLevelsRequested,
    D3D11_SDK_VERSION,
    &sd,
    &g_pSwapChain,
    &g_pd3dDevice,
    &FeatureLevelsSupported,
    &g_pImmediateContext)))
{
    return hr;
}
```

## ⓘ Note

If you request a `D3D_FEATURE_LEVEL_11_1` device on a computer with only the Direct3D 11.0 runtime, `D3D11CreateDeviceAndSwapChain` immediately exits with `E_INVALIDARG`. To safely request all possible feature levels on a computer with the DirectX 11.0 or DirectX 11.1 runtime, use this code:

```
const D3D_FEATURE_LEVEL lvl[] = { 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
};
UINT createDeviceFlags = 0;
#ifndef _DEBUG
createDeviceFlags |= D3D11_CREATE_DEVICE_DEBUG;
#endif

ID3D11Device* device = nullptr;
HRESULT hr = D3D11CreateDeviceAndSwapChain( nullptr,
D3D_DRIVER_TYPE_HARDWARE, nullptr, createDeviceFlags, lvl,
_countof(lvl), D3D11_SDK_VERSION, &sd, &g_pSwapChain, &g_pd3ddevice,
&FeatureLevelsSupported, &g_pImmediateContext );
if ( hr == E_INVALIDARG )
{
    hr = D3D11CreateDeviceAndSwapChain( nullptr,
D3D_DRIVER_TYPE_HARDWARE, nullptr, createDeviceFlags, &lvl[1],
_countof(lvl) - 1, D3D11_SDK_VERSION, &sd, &g_pSwapChain,
&g_pd3ddevice, &FeatureLevelsSupported, &g_pImmediateContext );
}

if (FAILED(hr))
return hr;
```

Create a render-target view by calling `ID3D11Device::CreateRenderTargetView` and bind the back-buffer as a render target by calling `ID3D11DeviceContext::OMSetRenderTargets`.

```
ID3D11Texture2D* pBackBuffer;
// Get a pointer to the back buffer
hr = g_pSwapChain->GetBuffer( 0, __uuidof( ID3D11Texture2D ), 
( LPVOID* )&pBackBuffer );

// Create a render-target view
g_pd3dDevice->CreateRenderTargetView( pBackBuffer, NULL,
&g_pRenderTargetView );

// Bind the view
g_pImmediateContext->OMSetRenderTargets( 1, &g_pRenderTargetView, NULL );
```

Create a viewport to define which parts of the render target will be visible. Define the viewport using the `D3D11_VIEWPORT` structure and set the viewport using the `ID3D11DeviceContext::RSSetViewports` method.

### C++

```
// Setup the viewport
D3D11_VIEWPORT vp;
vp.Width = 640;
vp.Height = 480;
vp.MinDepth = 0.0f;
vp.MaxDepth = 1.0f;
vp.TopLeftX = 0;
vp.TopLeftY = 0;
g_pImmediateContext->RSSetViewports( 1, &vp );
```

## Related topics

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Get the Device Feature Level

Article • 08/23/2019

This topic shows how to get the highest [feature level](#) supported by a [device](#). Direct3D 11 devices support a fixed set of feature levels that are defined in the [D3D\\_FEATURE\\_LEVEL](#) enumeration. When you know the highest [feature level](#) supported by a device, you can run code paths that are appropriate for that device.

## To get the device feature level

1. Call either the [D3D11CreateDevice](#) function or the [D3D11CreateDeviceAndSwapChain](#) functions while specifying `NULL` for the `ppDevice` parameter. You can do this before device creation.  
- or -  
Call [ID3D11Device::GetFeatureLevel](#) after device creation.
2. Examine the value of the returned [D3D\\_FEATURE\\_LEVEL](#) enumeration from the last step to determine the supported feature level.

The following code example demonstrates how to determine the highest supported feature level by calling the [D3D11CreateDevice](#) function. [D3D11CreateDevice](#) stores the highest supported feature level in the `FeatureLevel` variable. You can use this code to examine the value of the [D3D\\_FEATURE\\_LEVEL](#) enumerated type that [D3D11CreateDevice](#) returns. Note that this code lists all feature levels explicitly (for Direct3D 11.1 and Direct3D 11.2).

### Note

If the Direct3D 11.1 runtime is present on the computer and `pFeatureLevels` is set to `NULL`, this function won't create a [D3D\\_FEATURE\\_LEVEL\\_11\\_1](#) device. To create a [D3D\\_FEATURE\\_LEVEL\\_11\\_1](#) device, you must explicitly provide a [D3D\\_FEATURE\\_LEVEL](#) array that includes [D3D\\_FEATURE\\_LEVEL\\_11\\_1](#). If you provide a [D3D\\_FEATURE\\_LEVEL](#) array that contains [D3D\\_FEATURE\\_LEVEL\\_11\\_1](#) on a computer that doesn't have the Direct3D 11.1 runtime installed, this function immediately fails with `E_INVALIDARG`.

```
HRESULT hr = E_FAIL;
D3D_FEATURE_LEVEL MaxSupportedFeatureLevel = D3D_FEATURE_LEVEL_9_1;
D3D_FEATURE_LEVEL FeatureLevels[] = {
    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
};

hr = D3D11CreateDevice(
    NULL,
    D3D_DRIVER_TYPE_HARDWARE,
    NULL,
    0,
    &FeatureLevels,
    ARRSIZE(FeatureLevels),
    D3D11_SDK_VERSION,
    NULL,
    &MaxSupportedFeatureLevel,
    NULL
);

if(FAILED(hr))
{
    return hr;
}
```

The [10Level9 Reference](#) section lists the differences between how various [ID3D11Device](#) and [ID3D11DeviceContext](#) methods behave at various 10Level9 feature levels.

## Related topics

[Direct3D 11 on Downlevel Hardware](#)

[How to Use Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# How to: Create a Vertex Buffer

Article • 08/23/2019

[Vertex buffers](#) contain per vertex data. This topic shows how to initialize a static [vertex buffer](#), that is, a vertex buffer that does not change. For help initializing a non-static buffer, see the [remarks](#) section.

## To initialize a static vertex buffer

1. Define a structure that describes a vertex. For example, if your vertex data contains position data and color data, your structure would have one vector that describes the position and another that describes the color.
2. Allocate memory (using malloc or new) for the structure that you defined in step one. Fill this buffer with the actual vertex data that describes your geometry.
3. Create a buffer description by filling in a [D3D11\\_BUFFER\\_DESC](#) structure. Pass the D3D11\_BIND\_VERTEX\_BUFFER flag to the **BindFlags** member and pass the size of the structure from step one to the **ByteWidth** member.
4. Create a subresource data description by filling in a [D3D11\\_SUBRESOURCE\\_DATA](#) structure. The **pSysMem** member of the [D3D11\\_SUBRESOURCE\\_DATA](#) structure should point directly to the resource data created in step two.
5. Call [ID3D11Device::CreateBuffer](#) while passing the [D3D11\\_BUFFER\\_DESC](#) structure, the [D3D11\\_SUBRESOURCE\\_DATA](#) structure, and the address of a pointer to the [ID3D11Buffer](#) interface to initialize.

The following code example demonstrates how to create a vertex buffer. This example assumes that **g\_pd3dDevice** is a valid [ID3D11Device](#) object.

```
ID3D11Buffer*      g_pVertexBuffer;

// Define the data-type that
// describes a vertex.
struct SimpleVertexCombined
{
    XMFLOAT3 Pos;
    XMFLOAT3 Col;
};

// Supply the actual vertex data.
SimpleVertexCombined verticesCombo[] =
{
    XMFLOAT3( 0.0f, 0.5f, 0.5f ),
    XMFLOAT3( 0.0f, 0.0f, 0.5f ),
    XMFLOAT3( 0.5f, -0.5f, 0.5f ),
}
```

```

XMFLOAT3( 0.5f, 0.0f, 0.0f ),
XMFLOAT3( -0.5f, -0.5f, 0.5f ),
XMFLOAT3( 0.0f, 0.5f, 0.0f ),
};

// Fill in a buffer description.
D3D11_BUFFER_DESC bufferDesc;
bufferDesc.Usage          = D3D11_USAGE_DEFAULT;
bufferDesc.ByteWidth      = sizeof( SimpleVertexCombined ) * 3;
bufferDesc.BindFlags      = D3D11_BIND_VERTEX_BUFFER;
bufferDesc.CPUAccessFlags = 0;
bufferDesc.MiscFlags      = 0;

// Fill in the subresource data.
D3D11_SUBRESOURCE_DATA InitData;
InitData.pSysMem = verticesCombo;
InitData.SysMemPitch = 0;
InitData.SysMemSlicePitch = 0;

// Create the vertex buffer.
hr = g_pd3dDevice->CreateBuffer( &bufferDesc, &InitData, &g_pVertexBuffer );

```

## Remarks

Here are some ways to initialize a vertex buffer that changes over time.

1. Create a 2nd buffer with [D3D11\\_USAGE\\_STAGING](#); fill the second buffer using [ID3D11DeviceContext::Map](#), [ID3D11DeviceContext::Unmap](#); use [ID3D11DeviceContext::CopyResource](#) to copy from the staging buffer to the default buffer.
2. Use [ID3D11DeviceContext::UpdateSubresource](#) to copy data from memory.
3. Create a buffer with [D3D11\\_USAGE\\_DYNAMIC](#), and fill it with [ID3D11DeviceContext::Map](#), [ID3D11DeviceContext::Unmap](#) (using the Discard and NoOverwrite flags appropriately).

#1 and #2 are useful for content that changes less than once per frame. In general, GPU reads will be fast and CPU updates will be slower.

#3 is useful for content that changes more than once per frame. In general, GPU reads will be slower, but CPU updates will be faster.

## Related topics

[Buffers](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How to: Create an Index Buffer

Article • 08/23/2019

[Index buffers](#) contain index data. This topic shows how to initialize an [index buffer](#) in preparation for rendering.

## To initialize an index buffer

1. Create a buffer that contains your index information.
2. Create a buffer description by filling in a [D3D11\\_BUFFER\\_DESC](#) structure. Pass the `D3D11_BIND_INDEX_BUFFER` flag to the `BindFlags` member and pass the size of the buffer in bytes to the `ByteWidth` member.
3. Create a subresource data description by filling in a [D3D11\\_SUBRESOURCE\\_DATA](#) structure. The `pSysMem` member should point directly to the index data created in step one.
4. Call [ID3D11Device::CreateBuffer](#) while passing the [D3D11\\_BUFFER\\_DESC](#) structure, the [D3D11\\_SUBRESOURCE\\_DATA](#) structure, and the address of a pointer to the [ID3D11Buffer](#) interface to initialize.

The following code example demonstrates how to create an index buffer. This example assumes that

```
g_pd3dDevice
```

is a valid [ID3D11Device](#) object and that

```
g_pd3dContext
```

is a valid [ID3D11DeviceContext](#) object.

```
ID3D11Buffer *g_pIndexBuffer = NULL;  
  
// Create indices.  
unsigned int indices[] = { 0, 1, 2 };  
  
// Fill in a buffer description.  
D3D11_BUFFER_DESC bufferDesc;  
bufferDesc.Usage = D3D11_USAGE_DEFAULT;
```

```
bufferDesc.ByteWidth      = sizeof( unsigned int ) * 3;
bufferDesc.BindFlags       = D3D11_BIND_INDEX_BUFFER;
bufferDesc.CPUAccessFlags = 0;
bufferDesc.MiscFlags      = 0;

// Define the resource data.
D3D11_SUBRESOURCE_DATA InitData;
InitData.pSysMem = indices;
InitData.SysMemPitch = 0;
InitData.SysMemSlicePitch = 0;

// Create the buffer with the device.
hr = g_pd3dDevice->CreateBuffer( &bufferDesc, &InitData, &g_pIndexBuffer );
if( FAILED( hr ) )
    return hr;

// Set the buffer.
g_pd3dContext->IASetIndexBuffer( g_pIndexBuffer, DXGI_FORMAT_R32_UINT, 0 );
```

## Related topics

[Buffers](#)

[How to Use Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How to: Create a Constant Buffer

Article • 08/19/2020

Constant buffers contain shader constant data. This topic shows how to initialize a [constant buffer](#) in preparation for rendering.

## To initialize a constant buffer

1. Define a structure that describes the vertex shader constant data.
2. Allocate memory for the structure that you defined in step one. Fill this buffer with vertex shader constant data. You can use `malloc` or `new` to allocate the memory, or you can allocate memory for the structure from the stack.
3. Create a buffer description by filling in a [D3D11\\_BUFFER\\_DESC](#) structure. Pass the `D3D11_BIND_CONSTANT_BUFFER` flag to the `BindFlags` member and pass the size of the constant buffer description structure in bytes to the `ByteWidth` member.

### ! Note

The `D3D11_BIND_CONSTANT_BUFFER` flag cannot be combined with any other flags.

4. Create a subresource data description by filling in a [D3D11\\_SUBRESOURCE\\_DATA](#) structure. The `pSysMem` member of the [D3D11\\_SUBRESOURCE\\_DATA](#) structure must point directly to the vertex shader constant data that you created in step two.
5. Call [`ID3D11Device::CreateBuffer`](#) while passing the [D3D11\\_BUFFER\\_DESC](#) structure, the [D3D11\\_SUBRESOURCE\\_DATA](#) structure, and the address of a pointer to the [ID3D11Buffer](#) interface to initialize.

These code examples demonstrate how to create a constant buffer.

This example assumes that `g_pd3dDevice` is a valid [ID3D11Device](#) object and that `g_pd3dContext` is a valid [ID3D11DeviceContext](#) object.

C++

```
ID3D11Buffer* g_pConstantBuffer11 = NULL;  
  
// Define the constant data used to communicate with shaders.  
struct VS_CONSTANT_BUFFER
```

```

{
    XMFLOAT4X4 mWorldViewProj;
    XMFLOAT4 vSomeVectorThatMayBeNeededByASpecificShader;
    float fSomeFloatThatMayBeNeededByASpecificShader;
    float fTime;
    float fSomeFloatThatMayBeNeededByASpecificShader2;
    float fSomeFloatThatMayBeNeededByASpecificShader3;
} VS_CONSTANT_BUFFER;

// Supply the vertex shader constant data.
VS_CONSTANT_BUFFER VsConstData;
VsConstData.mWorldViewProj = {...};
VsConstData.vSomeVectorThatMayBeNeededByASpecificShader = XMFLOAT4(1,2,3,4);
VsConstData.fSomeFloatThatMayBeNeededByASpecificShader = 3.0f;
VsConstData.fTime = 1.0f;
VsConstData.fSomeFloatThatMayBeNeededByASpecificShader2 = 2.0f;
VsConstData.fSomeFloatThatMayBeNeededByASpecificShader3 = 4.0f;

// Fill in a buffer description.
D3D11_BUFFER_DESC cbDesc;
cbDesc.ByteWidth = sizeof( VS_CONSTANT_BUFFER );
cbDesc.Usage = D3D11_USAGE_DYNAMIC;
cbDesc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
cbDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
cbDesc.MiscFlags = 0;
cbDesc.StructureByteStride = 0;

// Fill in the subresource data.
D3D11_SUBRESOURCE_DATA InitData;
InitData.pSysMem = &VsConstData;
InitData.SysMemPitch = 0;
InitData.SysMemSlicePitch = 0;

// Create the buffer.
hr = g_pd3dDevice->CreateBuffer( &cbDesc, &InitData,
                                    &g_pConstantBuffer11 );

if( FAILED( hr ) )
    return hr;

// Set the buffer.
g_pd3dContext->VSSetConstantBuffers( 0, 1, &g_pConstantBuffer11 );

```

This example shows the associated HLSL cbuffer definition.

#### syntax

```

cbuffer VS_CONSTANT_BUFFER : register(b0)
{
    matrix mWorldViewProj;
    float4 vSomeVectorThatMayBeNeededByASpecificShader;
    float fSomeFloatThatMayBeNeededByASpecificShader;

```

```
    float fTime;
    float fSomeFloatThatMayBeNeededByASpecificShader2;
    float fSomeFloatThatMayBeNeededByASpecificShader3;
};
```

### ⓘ Note

Make sure to match the VS\_CONSTANT\_BUFFER memory layout in C++ with the HLSL layout. For info about how HLSL handles layout in memory, see [Packing Rules for Constant Variables](#).

## Related topics

[Buffers](#)

[How to Use Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How to: Create a Texture

Article • 08/23/2019

The simplest way to create a texture is to describe its properties and call the texture creation API. This topic shows how to create a texture.

## To create a texture

1. Fill in a [D3D11\\_TEXTURE2D\\_DESC](#) structure with a description of the texture parameters.
2. Create the texture by calling [ID3D11Device::CreateTexture2D](#) with the texture description.

This example creates a 256 x 256 texture, with [dynamic usage](#), for use as a [shader resource](#) with [cpu write access](#).

```
D3D11_TEXTURE2D_DESC desc;
desc.Width = 256;
desc.Height = 256;
desc.MipLevels = desc.ArraySize = 1;
desc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
desc.SampleDesc.Count = 1;
desc.Usage = D3D11_USAGE_DYNAMIC;
desc.BindFlags = D3D11_BIND_SHADER_RESOURCE;
desc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
desc.MiscFlags = 0;

ID3D11Device *pd3dDevice; // Don't forget to initialize this
ID3D11Texture2D *pTexture = NULL;
pd3dDevice->CreateTexture2D( &desc, NULL, &pTexture );
```

## Related topics

[How to Use Direct3D 11](#)

[Textures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# How to: Initialize a Texture Programmatically

Article • 08/23/2019

You can initialize a texture during object creation, or you can fill the object programmatically after it is created. This topic has several examples showing how to initialize textures that are created with different types of usages. This example assumes you already know how to [Create a Texture](#).

- [Default Usage](#)
- [Dynamic Usage](#)
- [Staging Usage](#)
- [Related topics](#)

## Default Usage

The most common type of usage is default usage. To fill a default texture (one created with `D3D11_USAGE_DEFAULT`) you can either:

- Call [`ID3D11Device::CreateTexture2D`](#) and initialize *pInitialData* to point to data provided from an application.
- or
- After calling [`ID3D11Device::CreateTexture2D`](#), use [`ID3D11DeviceContext::UpdateSubresource`](#) to fill the default texture with data from a pointer provided by the application.

## Dynamic Usage

To fill a dynamic texture (one created with `D3D11_USAGE_DYNAMIC`):

1. Get a pointer to the texture memory by passing in `D3D11_MAP_WRITE_DISCARD` when calling [`ID3D11DeviceContext::Map`](#).
2. Write data to the memory.
3. Call [`ID3D11DeviceContext::Unmap`](#) when you are finished writing data.

## Staging Usage

To fill a staging texture (one created with `D3D11_USAGE_STAGING`):

1. Get a pointer to the texture memory by passing in `D3D11_MAP_WRITE` when calling [ID3D11DeviceContext::Map](#).
2. Write data to the memory.
3. Call [ID3D11DeviceContext::Unmap](#) when you are finished writing data.

A staging texture can then be used as the source parameter to [ID3D11DeviceContext::CopyResource](#) or [ID3D11DeviceContext::CopySubresourceRegion](#) to fill a default or dynamic resource.

## Related topics

[How to Use Direct3D 11](#)

[Textures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How to: Initialize a Texture From a File

Article • 08/19/2020

You can use the [Windows Imaging Component](#) API to initialize a [texture](#) from a file. To load a texture, you must create a texture and a texture view. This topic shows how to use Windows Imaging Component (WIC) to create the texture and the view separately.

## ⓘ Note

This topic is useful for images that you create as simple 2D textures. For more complex resources, use [DDS](#). For a full-featured DDS file reader, writer, and texture processing pipeline, see [DirectXTex](#) and [DirectXTK](#).

At the end of this topic, you'll find the full example code. The topic describes the parts of the example code that create the texture and the view.

To initialize a texture and view separately.

1. Call [CoCreateInstance](#) to create the imaging factory interface ([IWICImagingFactory](#)).
2. Call the [IWICImagingFactory::CreateDecoderFromFilename](#) method to create a [IWICBitmapDecoder](#) object from an image file name.
3. Call the [IWICBitmapDecoder::GetFrame](#) method to retrieve the [IWICBitmapFrameDecode](#) interface for the frame of the image.
4. Call the [IWICBitmapSource::GetPixelFormat](#) method ([IWICBitmapFrameDecode](#) interface inherits from [IWICBitmapSource](#)) to get the pixel format of the image.
5. Convert the pixel format to a [DXGI\\_FORMAT](#) type according to this table:

WIC pixel format	Equivalent DXGI_FORMAT
GUID_WICPixelFormat128bppRGBAFloat	DXGI_FORMAT_R32G32B32A32_FLOAT
GUID_WICPixelFormat64bppRGBAHalf	DXGI_FORMAT_R16G16B16A16_FLOAT
GUID_WICPixelFormat64bppRGBA	DXGI_FORMAT_R16G16B16A16_UNORM
GUID_WICPixelFormat32bppRGBA	DXGI_FORMAT_R8G8B8A8_UNORM

<b>WIC pixel format</b>	<b>Equivalent DXGI_FORMAT</b>
GUID_WICPixelFormat32bppBGRA	DXGI_FORMAT_B8G8R8A8_UNORM (DXGI 1.1)
GUID_WICPixelFormat32bppBGR	DXGI_FORMAT_B8G8R8X8_UNORM (DXGI 1.1)
GUID_WICPixelFormat32bppRGBA1010102XR	DXGI_FORMAT_R10G10B10_XR_BIAS_A2_UNORM (DXGI 1.1)
GUID_WICPixelFormat32bppRGBA1010102	DXGI_FORMAT_R10G10B10A2_UNORM
GUID_WICPixelFormat32bppRGBE	DXGI_FORMAT_R9G9B9E5_SHAREDEXP
GUID_WICPixelFormat16bppBGRA5551	DXGI_FORMAT_B5G5R5A1_UNORM (DXGI 1.2)
GUID_WICPixelFormat16bppBGR565	DXGI_FORMAT_B5G6R5_UNORM (DXGI 1.2)
GUID_WICPixelFormat32bppGrayFloat	DXGI_FORMAT_R32_FLOAT*
GUID_WICPixelFormat16bppGrayHalf	DXGI_FORMAT_R16_FLOAT*
GUID_WICPixelFormat16bppGray	DXGI_FORMAT_R16_UNORM*
GUID_WICPixelFormat8bppGray	DXGI_FORMAT_R8_UNORM*
GUID_WICPixelFormat8bppAlpha	DXGI_FORMAT_A8_UNORM
GUID_WICPixelFormat96bppRGBFloat (Windows 8 WIC)	DXGI_FORMAT_R32G32B32_FLOAT

\* The single-channel DXGI formats are all red channel, so you need HLSL shader swizzles such as .rrr to render these as grayscale.

6. Call the [IWICBitmapSource::CopyPixels](#) method to copy the image pixels into a buffer. Use the [DXGI\\_FORMAT](#) type and the buffer to initialize the 2D texture resource and shader-resource-view object.
7. Call the [ID3D11Device::CreateTexture2D](#) method to initialize the 2D texture resource. In this call, pass the address of an [ID3D11Texture2D](#) interface pointer.

C++

```
// Create texture
D3D11_TEXTURE2D_DESC desc;
desc.Width = width;
desc.Height = height;
desc.MipLevels = 1;
desc.ArraySize = 1;
desc.Format = format;
```

```

desc.SampleDesc.Count = 1;
desc.SampleDesc.Quality = 0;
desc.Usage = D3D11_USAGE_DEFAULT;
desc.BindFlags = D3D11_BIND_SHADER_RESOURCE;
desc.CPUAccessFlags = 0;
desc.MiscFlags = 0;

D3D11_SUBRESOURCE_DATA initData;
initData.pSysMem = temp.get();
initData.SysMemPitch = static_cast<UINT>( rowPitch );
initData.SysMemSlicePitch = static_cast<UINT>( imageSize );

ID3D11Texture2D* tex = nullptr;
hr = d3dDevice->CreateTexture2D( &desc, &initData, &tex );

```

8. Call the [ID3D11Device::CreateShaderResourceView](#) method to initialize a shader-resource-view object. Pass either a **NULL** shader-resource-view description (to get a view with default parameters) or a non-**NULL** shader-resource-view description (to get a view with non-default parameters). If necessary, determine the texture type by calling [ID3D11Resource::GetType](#) and the texture format by calling [ID3D11ShaderResourceView::GetDesc](#).

C++

```

if ( SUCCEEDED(hr) && tex != 0 )
{
    if (textureView != 0)
    {
        D3D11_SHADER_RESOURCE_VIEW_DESC SRVDesc;
        memset( &SRVDesc, 0, sizeof( SRVDesc ) );
        SRVDesc.Format = format;
        SRVDesc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
        SRVDesc.Texture2D.MipLevels = 1;

        hr = d3dDevice->CreateShaderResourceView( tex, &SRVDesc,
textureView );
        if ( FAILED(hr) )
        {
            tex->Release();
            return hr;
        }
    }
}

```

The preceding example code assumes that the *d3dDevice* variable is an [ID3D11Device](#) object that has been previously initialized.

Here is the header that you can include in your app. The header declares the **CreateWICTextureFromFile** and **CreateWICTextureFromMemory** functions that you can call in your app to create a texture from a file and from memory.

```

d3dContext,
                           _In_bytector(_wicDataSize) const
uint8_t* wicData,
                           _In_ size_t wicDataSize,
                           _Out_opt_ ID3D11Resource** texture,
                           _Out_opt_ ID3D11ShaderResourceView**
textureView,
                           _In_ size_t maxsize = 0
);

HRESULT CreateWICTextureFromFile( _In_ ID3D11Device* d3dDevice,
                           _In_opt_ ID3D11DeviceContext* d3dContext,
                           _In_z_ const wchar_t* szFileName,
                           _Out_opt_ ID3D11Resource** texture,
                           _Out_opt_ ID3D11ShaderResourceView** textureView,
                           _In_ size_t maxsize = 0
);

```

Here is the full source that you can use in your app. The source implements the `CreateWICTextureFromFile` and `CreateWICTextureFromMemory` functions.

C++

```

//-----
// File: WICTextureLoader.cpp
//
// Function for loading a WIC image and creating a Direct3D 11 runtime
// texture for it
// (auto-generating mipmaps if possible)
//
// Note: Assumes application has already called CoInitializeEx
//
// Warning: CreateWICTexture* functions are not thread-safe if given a
// d3dContext instance for
//           auto-gen mipmap support.
//
// Note these functions are useful for images created as simple 2D textures.
For
// more complex resources, DDSTextureLoader is an excellent light-weight
// runtime loader.
// For a full-featured DDS file reader, writer, and texture processing
// pipeline see
// the 'Texconv' sample and the 'DirectXTex' library.
//
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF
// ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
// PARTICULAR PURPOSE.
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//

```

```
// https://go.microsoft.com/fwlink/?LinkId=248926
// https://go.microsoft.com/fwlink/?LinkId=248929
//-----
-----

// We could load multi-frame images (TIFF/GIF) into a texture array.
// For now, we just load the first frame (note: DirectXTex supports multi-
frame images)

#include <dxgiformat.h>
#include <assert.h>

#pragma warning(push)
#pragma warning(disable : 4005)
#include <wincodec.h>
#pragma warning(pop)

#include <memory>

#include "WICTextureLoader.h"

#if (_WIN32_WINNT >= 0x0602 /*_WIN32_WINNT_WIN8*/) &&
!defined(DXGI_1_2_FORMATS)
#define DXGI_1_2_FORMATS
#endif

//-----
-----

template<class T> class ScopedObject
{
public:
    explicit ScopedObject( T *p = 0 ) : _pointer(p) {}
    ~ScopedObject()
    {
        if ( _pointer )
        {
            _pointer->Release();
            _pointer = nullptr;
        }
    }

    bool IsNull() const { return (!_pointer); }

    T& operator*() { return *_pointer; }
    T* operator->() { return _pointer; }
    T** operator&() { return &_pointer; }

    void Reset(T *p = 0) { if ( _pointer ) { _pointer->Release(); } _pointer
= p; }

    T* Get() const { return _pointer; }

private:
    ScopedObject(const ScopedObject&);
    ScopedObject& operator=(const ScopedObject&);
```

```

    T* _pointer;
};

//-----
// WIC Pixel Format Translation Data
//-----

struct WICTranslate
{
    GUID           wic;
    DXGI_FORMAT    format;
};

static WICTranslate g_WICFormats[] =
{
    { GUID_WICPixelFormat128bppRGBAFloat,
DXGI_FORMAT_R32G32B32A32_FLOAT },

    { GUID_WICPixelFormat64bppRGBAHalf,
DXGI_FORMAT_R16G16B16A16_FLOAT },
    { GUID_WICPixelFormat64bppRGBA,
DXGI_FORMAT_R16G16B16A16_UNORM },

    { GUID_WICPixelFormat32bppRGB,
DXGI_FORMAT_R8G8B8A8_UNORM },
    { GUID_WICPixelFormat32bppBGRA,
DXGI_FORMAT_B8G8R8A8_UNORM },
    { GUID_WICPixelFormat32bppBGR,
DXGI_FORMAT_B8G8R8X8_UNORM },
    { GUID_WICPixelFormat32bppRGB1010102XR,
DXGI_FORMAT_R10G10B10_XR_BIAS_A2_UNORM }, // DXGI 1.1
    { GUID_WICPixelFormat32bppRGB1010102,
DXGI_FORMAT_R10G10B10A2_UNORM },
    { GUID_WICPixelFormat32bppRGBE,
DXGI_FORMAT_R9G9B9E5_SHAREDEXP },

#endif // DXGI_1_2_FORMATS

    { GUID_WICPixelFormat16bppBGRA5551,
DXGI_FORMAT_B5G5R5A1_UNORM },
    { GUID_WICPixelFormat16bppBGR565,
DXGI_FORMAT_B5G6R5_UNORM },

#endif // DXGI_1_2_FORMATS

    { GUID_WICPixelFormat32bppGrayFloat,
DXGI_FORMAT_R32_FLOAT },
    { GUID_WICPixelFormat16bppGrayHalf,
DXGI_FORMAT_R16_FLOAT },
    { GUID_WICPixelFormat16bppGray,
DXGI_FORMAT_R16_UNORM },
    { GUID_WICPixelFormat8bppGray,
DXGI_FORMAT_R8_UNORM },

    { GUID_WICPixelFormat8bppAlpha,
DXGI_FORMAT_A8_UNORM },

#if (_WIN32_WINNT >= 0x0602 /*_WIN32_WINNT_WIN8*/)

```

```

        { GUID_WICPixelFormat96bppRGBFLOAT,           DXGI_FORMAT_R32G32B32_FLOAT
},
#endif
};

//-----
// WIC Pixel Format nearest conversion table
//-----
//-----


struct WICConvert
{
    GUID source;
    GUID target;
};

static WICConvert g_WICConvert[] =
{
    // Note target GUID in this conversion table must be one of those
    // directly supported formats (above).

    { GUID_WICPixelFormatBlackWhite,               GUID_WICPixelFormat8bppGray
}, // DXGI_FORMAT_R8_UNORM

    { GUID_WICPixelFormat1bppIndexed,             DXGI_FORMAT_R8G8B8A8_UNORM
}, // DXGI_FORMAT_R8G8B8A8_UNORM
    { GUID_WICPixelFormat2bppIndexed,             DXGI_FORMAT_R8G8B8A8_UNORM
}, // DXGI_FORMAT_R8G8B8A8_UNORM
    { GUID_WICPixelFormat4bppIndexed,             DXGI_FORMAT_R8G8B8A8_UNORM
}, // DXGI_FORMAT_R8G8B8A8_UNORM
    { GUID_WICPixelFormat8bppIndexed,             DXGI_FORMAT_R8G8B8A8_UNORM
}, // DXGI_FORMAT_R8G8B8A8_UNORM

    { GUID_WICPixelFormat2bppGray,                DXGI_FORMAT_R8_UNORM
}, // DXGI_FORMAT_R8_UNORM
    { GUID_WICPixelFormat4bppGray,                DXGI_FORMAT_R8_UNORM
}, // DXGI_FORMAT_R8_UNORM

    { GUID_WICPixelFormat16bppGrayFixedPoint,
GUID_WICPixelFormat16bppGrayHalf }, // DXGI_FORMAT_R16_FLOAT
    { GUID_WICPixelFormat32bppGrayFixedPoint,
GUID_WICPixelFormat32bppGrayFloat }, // DXGI_FORMAT_R32_FLOAT

#ifdef DXGI_1_2_FORMATS

    { GUID_WICPixelFormat16bppBGR555,
GUID_WICPixelFormat16bppBGRA5551 }, // DXGI_FORMAT_B5G5R5A1_UNORM

#else

    { GUID_WICPixelFormat16bppBGR555,             DXGI_FORMAT_R8G8B8A8_UNORM
}, // DXGI_FORMAT_R8G8B8A8_UNORM
    { GUID_WICPixelFormat16bppBGRA5551,            DXGI_FORMAT_R8G8B8A8_UNORM
}, // DXGI_FORMAT_R8G8B8A8_UNORM

```

```

        { GUID_WICPixelFormat16bppBGR565,           GUID_WICPixelFormat32bppRGBA
}, // DXGI_FORMAT_R8G8B8A8_UNORM

#endif // DXGI_1_2_FORMATS

        { GUID_WICPixelFormat32bppBGR101010,
GUID_WICPixelFormat32bppRGBA101010 }, // DXGI_FORMAT_R10G10B10A2_UNORM

        { GUID_WICPixelFormat24bppBGR,                 GUID_WICPixelFormat32bppRGBA
}, // DXGI_FORMAT_R8G8B8A8_UNORM
        { GUID_WICPixelFormat24bppRGB,                GUID_WICPixelFormat32bppRGBA
}, // DXGI_FORMAT_R8G8B8A8_UNORM
        { GUID_WICPixelFormat32bppPBGRA,              GUID_WICPixelFormat32bppRGBA
}, // DXGI_FORMAT_R8G8B8A8_UNORM
        { GUID_WICPixelFormat32bppPRGBA,              GUID_WICPixelFormat32bppRGBA
}, // DXGI_FORMAT_R8G8B8A8_UNORM

        { GUID_WICPixelFormat48bppRGB,                GUID_WICPixelFormat64bppRGBA
}, // DXGI_FORMAT_R16G16B16A16_UNORM
        { GUID_WICPixelFormat48bppBGR,                GUID_WICPixelFormat64bppRGBA
}, // DXGI_FORMAT_R16G16B16A16_UNORM
        { GUID_WICPixelFormat64bppBGRA,               GUID_WICPixelFormat64bppRGBA
}, // DXGI_FORMAT_R16G16B16A16_UNORM
        { GUID_WICPixelFormat64bppPRGBA,              GUID_WICPixelFormat64bppRGBA
}, // DXGI_FORMAT_R16G16B16A16_UNORM
        { GUID_WICPixelFormat64bppPBGRA,              GUID_WICPixelFormat64bppRGBA
}, // DXGI_FORMAT_R16G16B16A16_UNORM

        { GUID_WICPixelFormat48bppRGBFixedPoint,
GUID_WICPixelFormat64bppRGBAHalf }, // DXGI_FORMAT_R16G16B16A16_FLOAT
        { GUID_WICPixelFormat48bppBGRFixedPoint,
GUID_WICPixelFormat64bppRGBAHalf }, // DXGI_FORMAT_R16G16B16A16_FLOAT
        { GUID_WICPixelFormat64bppRGBAFixedPoint,
GUID_WICPixelFormat64bppRGBAHalf }, // DXGI_FORMAT_R16G16B16A16_FLOAT
        { GUID_WICPixelFormat64bppBGRAFixedPoint,
GUID_WICPixelFormat64bppRGBAHalf }, // DXGI_FORMAT_R16G16B16A16_FLOAT
        { GUID_WICPixelFormat64bppRGBFixedPoint,
GUID_WICPixelFormat64bppRGBAHalf }, // DXGI_FORMAT_R16G16B16A16_FLOAT
        { GUID_WICPixelFormat64bppRGBHalf,
GUID_WICPixelFormat64bppRGBAHalf }, // DXGI_FORMAT_R16G16B16A16_FLOAT
        { GUID_WICPixelFormat48bppRGBHalf,
GUID_WICPixelFormat64bppRGBAHalf }, // DXGI_FORMAT_R16G16B16A16_FLOAT

        { GUID_WICPixelFormat96bppRGBFixedPoint,
GUID_WICPixelFormat128bppRGBAFloat }, // DXGI_FORMAT_R32G32B32A32_FLOAT
        { GUID_WICPixelFormat128bppPRGBAFloat,
GUID_WICPixelFormat128bppRGBAFloat }, // DXGI_FORMAT_R32G32B32A32_FLOAT
        { GUID_WICPixelFormat128bppRGBFloat,
GUID_WICPixelFormat128bppRGBAFloat }, // DXGI_FORMAT_R32G32B32A32_FLOAT
        { GUID_WICPixelFormat128bppRGBAFixedPoint,
GUID_WICPixelFormat128bppRGBAFloat }, // DXGI_FORMAT_R32G32B32A32_FLOAT
        { GUID_WICPixelFormat128bppRGBFixedPoint,
GUID_WICPixelFormat128bppRGBAFloat }, // DXGI_FORMAT_R32G32B32A32_FLOAT
        { GUID_WICPixelFormat128bppRGBAFloat },
GUID_WICPixelFormat128bppRGBAFixedPoint }, // DXGI_FORMAT_R32G32B32A32_FLOAT

        { GUID_WICPixelFormat32bppCMYK,                GUID_WICPixelFormat32bppRGBA
}

```

```

        }, // DXGI_FORMAT_R8G8B8A8_UNORM
        { GUID_WICPixelFormat64bppCMYK,                      GUID_WICPixelFormat64bppRGBA
        }, // DXGI_FORMAT_R16G16B16A16_UNORM
        { GUID_WICPixelFormat40bppCMYKAlpha,                 GUID_WICPixelFormat64bppRGBA
        }, // DXGI_FORMAT_R16G16B16A16_UNORM
        { GUID_WICPixelFormat80bppCMYKAlpha,                 GUID_WICPixelFormat64bppRGBA
        }, // DXGI_FORMAT_R16G16B16A16_UNORM

#if (_WIN32_WINNT >= 0x0602 /*_WIN32_WINNT_WIN8*/)
    { GUID_WICPixelFormat32bppRGB,                      GUID_WICPixelFormat32bppRGBA
    }, // DXGI_FORMAT_R8G8B8A8_UNORM
    { GUID_WICPixelFormat64bppRGB,                      GUID_WICPixelFormat64bppRGBA
    }, // DXGI_FORMAT_R16G16B16A16_UNORM
    { GUID_WICPixelFormat64bppPRGBAHalf,
GUID_WICPixelFormat64bppRGBAHalf }, // DXGI_FORMAT_R16G16B16A16_FLOAT
#endif

        // We don't support n-channel formats
};

//-----
-----

static IWICImagingFactory* _GetWIC()
{
    static IWICImagingFactory* s_Factory = nullptr;

    if ( s_Factory )
        return s_Factory;

    HRESULT hr = CoCreateInstance(
        CLSID_WICImagingFactory,
        nullptr,
        CLSCTX_INPROC_SERVER,
        __uuidof(IWICImagingFactory),
        (LPVOID*)&s_Factory
    );

    if ( FAILED(hr) )
    {
        s_Factory = nullptr;
        return nullptr;
    }

    return s_Factory;
}

//-----
-----

static DXGI_FORMAT _WICToDXGI( const GUID& guid )
{
    for( size_t i=0; i < _countof(g_WICFormats); ++i )
    {
        if ( memcmp( &g_WICFormats[i].wic, &guid, sizeof(GUID) ) == 0 )
            return g_WICFormats[i].format;
    }
}

```

```

        return DXGI_FORMAT_UNKNOWN;
    }

//-----
-----

static size_t _WICBitsPerPixel( REFGUID targetGuid )
{
    IWICImagingFactory* pWIC = _GetWIC();
    if ( !pWIC )
        return 0;

    ScopedObject<IWICComponentInfo> cinfo;
    if ( FAILED( pWIC->CreateComponentInfo( targetGuid, &cinfo ) ) )
        return 0;

    WICComponentType type;
    if ( FAILED( cinfo->GetComponentType( &type ) ) )
        return 0;

    if ( type != WICPixelFormat )
        return 0;

    ScopedObject<IWICPixelFormatInfo> pfinfo;
    if ( FAILED( cinfo->QueryInterface( __uuidof(IWICPixelFormatInfo),
reinterpret_cast<void**>( &pfinfo ) ) ) )
        return 0;

    UINT bpp;
    if ( FAILED( pfinfo->GetBitsPerPixel( &bpp ) ) )
        return 0;

    return bpp;
}

//-----
-----

static HRESULT CreateTextureFromWIC( _In_ ID3D11Device* d3dDevice,
                                    _In_opt_ ID3D11DeviceContext*
d3dContext,
                                    _In_ IWICBitmapFrameDecode *frame,
                                    _Out_opt_ ID3D11Resource** texture,
                                    _Out_opt_ ID3D11ShaderResourceView**
textureView,
                                    _In_ size_t maxsize )
{
    UINT width, height;
    HRESULT hr = frame->GetSize( &width, &height );
    if ( FAILED(hr) )
        return hr;

    assert( width > 0 && height > 0 );

    if ( !maxsize )
    {

```

```

        // This is a bit conservative because the hardware could support
        larger textures than
            // the Feature Level defined minimums, but doing it this way is much
            easier and more
                // performant for WIC than the 'fail and retry' model used by
                DDSTextureLoader

        switch( d3dDevice->GetFeatureLevel() )
        {
            case D3D_FEATURE_LEVEL_9_1:
            case D3D_FEATURE_LEVEL_9_2:
                maxsize = 2048 /*D3D_FL9_1_REQ_TEXTURE2D_U_OR_V_DIMENSION*/;
                break;

            case D3D_FEATURE_LEVEL_9_3:
                maxsize = 4096 /*D3D_FL9_3_REQ_TEXTURE2D_U_OR_V_DIMENSION*/;
                break;

            case D3D_FEATURE_LEVEL_10_0:
            case D3D_FEATURE_LEVEL_10_1:
                maxsize = 8192 /*D3D10_REQ_TEXTURE2D_U_OR_V_DIMENSION*/;
                break;

            default:
                maxsize = D3D11_REQ_TEXTURE2D_U_OR_V_DIMENSION;
                break;
        }

    }

assert( maxsize > 0 );

UINT twidth, theight;
if ( width > maxsize || height > maxsize )
{
    float ar = static_cast<float>(height) / static_cast<float>(width);
    if ( width > height )
    {
        twidth = static_cast<UINT>( maxsize );
        theight = static_cast<UINT>( static_cast<float>(maxsize) * ar );
    }
    else
    {
        theight = static_cast<UINT>( maxsize );
        twidth = static_cast<UINT>( static_cast<float>(maxsize) / ar );
    }
    assert( twidth <= maxsize && theight <= maxsize );
}
else
{
    twidth = width;
    theight = height;
}

// Determine format
WICPixelFormatGUID pixelFormat;

```

```

hr = frame->GetPixelFormat( &pixelFormat );
if ( FAILED(hr) )
    return hr;

WICPixelFormatGUID convertGUID;
memcpy( &convertGUID, &pixelFormat, sizeof(WICPixelFormatGUID) );

size_t bpp = 0;

DXGI_FORMAT format = _WICToDXGI( pixelFormat );
if ( format == DXGI_FORMAT_UNKNOWN )
{
    for( size_t i=0; i < _countof(g_WICConvert); ++i )
    {
        if ( memcmp( &g_WICConvert[i].source, &pixelFormat,
sizeof(WICPixelFormatGUID) ) == 0 )
        {
            memcpy( &convertGUID, &g_WICConvert[i].target,
sizeof(WICPixelFormatGUID) );

            format = _WICToDXGI( g_WICConvert[i].target );
            assert( format != DXGI_FORMAT_UNKNOWN );
            bpp = _WICBitsPerPixel( convertGUID );
            break;
        }
    }

    if ( format == DXGI_FORMAT_UNKNOWN )
        return HRESULT_FROM_WIN32( ERROR_NOT_SUPPORTED );
}
else
{
    bpp = _WICBitsPerPixel( pixelFormat );
}

if ( !bpp )
    return E_FAIL;

// Verify our target format is supported by the current device
// (handles WDDM 1.0 or WDDM 1.1 device driver cases as well as DirectX
11.0 Runtime without 16bpp format support)
UINT support = 0;
hr = d3dDevice->CheckFormatSupport( format, &support );
if ( FAILED(hr) || !(support & D3D11_FORMAT_SUPPORT_TEXTURE2D) )
{
    // Fallback to RGBA 32-bit format which is supported by all devices
    memcpy( &convertGUID, &GUID_WICPixelFormat32bppRGBA,
sizeof(WICPixelFormatGUID) );
    format = DXGI_FORMAT_R8G8B8A8_UNORM;
    bpp = 32;
}

// Allocate temporary memory for image
size_t rowPitch = ( twidth * bpp + 7 ) / 8;
size_t imageSize = rowPitch * theight;

```

```

    std::unique_ptr<uint8_t[]> temp( new uint8_t[ imageSize ] );

    // Load image data
    if ( memcmp( &convertGUID, &pixelFormat, sizeof(GUID) ) == 0
        && twidth == width
        && theight == height )
    {
        // No format conversion or resize needed
        hr = frame->CopyPixels( 0, static_cast<UINT>( rowPitch ),
static_cast<UINT>( imageSize ), temp.get() );
        if ( FAILED(hr) )
            return hr;
    }
    else if ( twidth != width || theight != height )
    {
        // Resize
        IWICImagingFactory* pWIC = _GetWIC();
        if ( !pWIC )
            return E_NOINTERFACE;

        ScopedObject<IWICBitmapScaler> scaler;
        hr = pWIC->CreateBitmapScaler( &scaler );
        if ( FAILED(hr) )
            return hr;

        hr = scaler->Initialize( frame, twidth, theight,
WICBitmapInterpolationModeFant );
        if ( FAILED(hr) )
            return hr;

        WICPixelFormatGUID pfScaler;
        hr = scaler->GetPixelFormat( &pfScaler );
        if ( FAILED(hr) )
            return hr;

        if ( memcmp( &convertGUID, &pfScaler, sizeof(GUID) ) == 0 )
        {
            // No format conversion needed
            hr = scaler->CopyPixels( 0, static_cast<UINT>( rowPitch ),
static_cast<UINT>( imageSize ), temp.get() );
            if ( FAILED(hr) )
                return hr;
        }
        else
        {
            ScopedObject<IWICFormatConverter> FC;
            hr = pWIC->CreateFormatConverter( &FC );
            if ( FAILED(hr) )
                return hr;

            hr = FC->Initialize( scaler.Get(), convertGUID,
WICBitmapDitherTypeErrorDiffusion, 0, 0, WICBitmapPaletteTypeCustom );
            if ( FAILED(hr) )
                return hr;
        }
    }
}

```

```

        hr = FC->CopyPixels( 0, static_cast<UINT>( rowPitch ),
static_cast<UINT>( imageSize ), temp.get() );
        if ( FAILED(hr) )
            return hr;
    }
}
else
{
    // Format conversion but no resize
    IWICImagingFactory* pWIC = _GetWIC();
    if ( !pWIC )
        return E_NOINTERFACE;

    ScopedObject<IWICFormatConverter> FC;
    hr = pWIC->CreateFormatConverter( &FC );
    if ( FAILED(hr) )
        return hr;

    hr = FC->Initialize( frame, convertGUID,
WICBitmapDitherTypeErrorDiffusion, 0, 0, WICBitmapPaletteTypeCustom );
    if ( FAILED(hr) )
        return hr;

    hr = FC->CopyPixels( 0, static_cast<UINT>( rowPitch ),
static_cast<UINT>( imageSize ), temp.get() );
    if ( FAILED(hr) )
        return hr;
}

// See if format is supported for auto-gen mipmaps (varies by feature
level)
bool autogen = false;
if ( d3dContext != 0 && textureView != 0 ) // Must have context and
shader-view to auto generate mipmaps
{
    UINT fmtSupport = 0;
    hr = d3dDevice->CheckFormatSupport( format, &fmtSupport );
    if ( SUCCEEDED(hr) && ( fmtSupport &
D3D11_FORMAT_SUPPORT_MIP_AUTOGEN ) )
    {
        autogen = true;
    }
}

// Create texture
D3D11_TEXTURE2D_DESC desc;
desc.Width = twidth;
desc.Height = theight;
desc.MipLevels = (autogen) ? 0 : 1;
desc.ArraySize = 1;
desc.Format = format;
desc.SampleDesc.Count = 1;
desc.SampleDesc.Quality = 0;
desc.Usage = D3D11_USAGE_DEFAULT;

```

```

desc.BindFlags = (autogen) ? (D3D11_BIND_SHADER_RESOURCE | D3D11_BIND_RENDER_TARGET) : (D3D11_BIND_SHADER_RESOURCE);
desc.CPUAccessFlags = 0;
desc.MiscFlags = (autogen) ? D3D11_RESOURCE_MISC_GENERATE_MIPS : 0;

D3D11_SUBRESOURCE_DATA initData;
initData.pSysMem = temp.get();
initData.SysMemPitch = static_cast<UINT>( rowPitch );
initData.SysMemSlicePitch = static_cast<UINT>( imageSize );

ID3D11Texture2D* tex = nullptr;
hr = d3dDevice->CreateTexture2D( &desc, (autogen) ? nullptr : &initData,
&tex );
if ( SUCCEEDED(hr) && tex != 0 )
{
    if (textureView != 0)
    {
        D3D11_SHADER_RESOURCE_VIEW_DESC SRVDesc;
        memset( &SRVDesc, 0, sizeof( SRVDesc ) );
        SRVDesc.Format = format;
        SRVDesc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
        SRVDesc.Texture2D.MipLevels = (autogen) ? -1 : 1;

        hr = d3dDevice->CreateShaderResourceView( tex, &SRVDesc,
textureView );
        if ( FAILED(hr) )
        {
            tex->Release();
            return hr;
        }

        if ( autogen )
        {
            assert( d3dContext != 0 );
            d3dContext->UpdateSubresource( tex, 0, nullptr, temp.get(),
static_cast<UINT>(rowPitch), static_cast<UINT>(imageSize) );
            d3dContext->GenerateMips( *textureView );
        }
    }

    if (texture != 0)
    {
        *texture = tex;
    }
    else
    {
#ifndef _DEBUG || !defined(PROFILE)
        tex->SetPrivateData( WKPDID_D3DDescriptorName,
                           sizeof("WICTextureLoader")-1,
                           "WICTextureLoader"
                           );
#endif
        tex->Release();
    }
}

```

```

        return hr;
    }

//-----
HRESULT CreateWICTextureFromMemory( _In_ ID3D11Device* d3dDevice,
                                    _In_opt_ ID3D11DeviceContext*
d3dContext,
                                    _In_bytecount_(wicDataSize) const
uint8_t* wicData,
                                    _In_ size_t wicDataSize,
                                    _Out_opt_ ID3D11Resource** texture,
                                    _Out_opt_ ID3D11ShaderResourceView**
textureView,
                                    _In_ size_t maxsize
)
{
    if (!d3dDevice || !wicData || (!texture && !textureView))
    {
        return E_INVALIDARG;
    }

    if ( !wicDataSize )
    {
        return E_FAIL;
    }

#ifndef _M_AMD64
    if ( wicDataSize > 0xFFFFFFFF )
        return HRESULT_FROM_WIN32( ERROR_FILE_TOO_LARGE );
#endif

IWICImagingFactory* pWIC = _GetWIC();
if ( !pWIC )
    return E_NOINTERFACE;

// Create input stream for memory
ScopedObject<IWICStream> stream;
HRESULT hr = pWIC->CreateStream( &stream );
if ( FAILED(hr) )
    return hr;

hr = stream->InitializeFromMemory( const_cast<uint8_t*>( wicData ),
static_cast<DWORD>( wicDataSize ) );
if ( FAILED(hr) )
    return hr;

// Initialize WIC
ScopedObject<IWICBitmapDecoder> decoder;
hr = pWIC->CreateDecoderFromStream( stream.Get(), 0,
WICDecodeMetadataCacheOnDemand, &decoder );
if ( FAILED(hr) )
    return hr;

```

```

    ScopedObject<IWICBitmapFrameDecode> frame;
    hr = decoder->GetFrame( 0, &frame );
    if ( FAILED(hr) )
        return hr;

    hr = CreateTextureFromWIC( d3dDevice, d3dContext, frame.Get(), texture,
    textureView, maxsize );
    if ( FAILED(hr) )
        return hr;

#ifndef _DEBUG_ || _PROFILE_
    if (texture != 0 && *texture != 0)
    {
        (*texture)->SetPrivateData( WKPID_D3DDescriptorName,
                                      sizeof("WICTextureLoader")-1,
                                      "WICTextureLoader"
                                    );
    }

    if (textureView != 0 && *textureView != 0)
    {
        (*textureView)->SetPrivateData( WKPID_D3DDescriptorName,
                                         sizeof("WICTextureLoader")-1,
                                         "WICTextureLoader"
                                       );
    }
#endif

    return hr;
}

//-----
-----

HRESULT CreateWICTextureFromFile( _In_ ID3D11Device* d3dDevice,
                                  _In_opt_ ID3D11DeviceContext* d3dContext,
                                  _In_z_ const wchar_t* fileName,
                                  _Out_opt_ ID3D11Resource** texture,
                                  _Out_opt_ ID3D11ShaderResourceView** textureView,
                                  _In_ size_t maxsize )
{
    if (!d3dDevice || !fileName || (!texture && !textureView))
    {
        return E_INVALIDARG;
    }

    IWICImagingFactory* pWIC = _GetWIC();
    if ( !pWIC )
        return E_NOINTERFACE;

    // Initialize WIC
    ScopedObject<IWICBitmapDecoder> decoder;
    HRESULT hr = pWIC->CreateDecoderFromFilename( fileName, 0, GENERIC_READ,
    WICDecodeMetadataCacheOnDemand, &decoder );
    if ( FAILED(hr) )

```

```

    return hr;

ScopedObject<IWICBitmapFrameDecode> frame;
hr = decoder->GetFrame( 0, &frame );
if ( FAILED(hr) )
    return hr;

hr = CreateTextureFromWIC( d3dDevice, d3dContext, frame.Get(), texture,
textureView, maxsize );
if ( FAILED(hr) )
    return hr;

#if defined(_DEBUG) || defined(PROFILE)
if (texture != 0 || textureView != 0)
{
    CHAR strFileA[MAX_PATH];
    WideCharToMultiByte( CP_ACP,
                        WC_NO_BEST_FIT_CHARS,
                        fileName,
                        -1,
                        strFileA,
                        MAX_PATH,
                        nullptr,
                        FALSE
                    );
    const CHAR* pstrName = strrchr( strFileA, '\\' );
    if (!pstrName)
    {
        pstrName = strFileA;
    }
    else
    {
        pstrName++;
    }

    if (texture != 0 && *texture != 0)
    {
        (*texture)->SetPrivateData( WKPID_D3DDescribeObjectName,
                                    static_cast<UINT>(
            strlen_s(pstrName, MAX_PATH) ),
                                    pstrName
                                );
    }

    if (textureView != 0 && *textureView != 0 )
    {
        (*textureView)->SetPrivateData( WKPID_D3DDescribeObjectName,
                                    static_cast<UINT>(
            strlen_s(pstrName, MAX_PATH) ),
                                    pstrName
                                );
    }
}
#endif

```

```
    return hr;  
}
```

## Related topics

[How to Use Direct3D 11](#)

[Textures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How to: Use dynamic resources

Article • 08/23/2019

## Important APIs

- [ID3D11DeviceContext::Map](#)
- [ID3D11DeviceContext::Unmap](#)
- [D3D11\\_USAGE](#)

You create and use dynamic resources when your app needs to change data in those resources. You can create textures and buffers for dynamic usage.

## What you need to know

### Technologies

- [How to: Create a Texture](#)
- [How to: Initialize a Texture Programmatically](#)
- [How to: Create a Constant Buffer](#)

### Prerequisites

We assume that you are familiar with C++. You also need basic experience with graphics programming concepts.

## Instructions

### Step 1: Specify dynamic usage

If you want your app to be able to make changes to resources, you must specify those resources as dynamic and writable when you create them.

#### To specify dynamic usage

1. Specify the resource usage as dynamic. For example, specify the [D3D11\\_USAGE\\_DYNAMIC](#) value in the [Usage](#) member of [D3D11\\_BUFFER\\_DESC](#) for a vertex or constant buffer and specify the [D3D11\\_USAGE\\_DYNAMIC](#) value in the [Usage](#) member of [D3D11\\_TEXTURE2D\\_DESC](#) for a 2D texture.

2. Specify the resource as writable. For example, specify the [D3D11\\_CPU\\_ACCESS\\_WRITE](#) value in the `CPUAccessFlags` member of [D3D11\\_BUFFER\\_DESC](#) or [D3D11\\_TEXTURE2D\\_DESC](#).
3. Pass the resource description to the creation function. For example, pass the address of [D3D11\\_BUFFER\\_DESC](#) to [ID3D11Device::CreateBuffer](#) and pass the address of [D3D11\\_TEXTURE2D\\_DESC](#) to [ID3D11Device::CreateTexture2D](#).

Here is an example of creating a dynamic vertex buffer:

C++

```
// Create a vertex buffer for a triangle.

float2 triangleVertices[] =
{
    float2(-0.5f, -0.5f),
    float2(0.0f, 0.5f),
    float2(0.5f, -0.5f),
};

D3D11_BUFFER_DESC vertexBufferDesc = { 0 };
vertexBufferDesc.ByteWidth = sizeof\(float2\)\*
ARRAYSIZE\(triangleVertices\);
vertexBufferDesc.Usage = D3D11_USAGE_DYNAMIC;
vertexBufferDesc.BindFlags = D3D11_BIND_VERTEX_BUFFER;
vertexBufferDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
vertexBufferDesc.MiscFlags = 0;
vertexBufferDesc.StructureByteStride = 0;

D3D11_SUBRESOURCE_DATA vertexBufferData;
vertexBufferData.pSysMem = triangleVertices;
vertexBufferData.SysMemPitch = 0;
vertexBufferData.SysMemSlicePitch = 0;

DX::ThrowIfFailed(
    m_d3dDevice->CreateBuffer(
        &vertexBufferDesc,
        &vertexBufferData,
        &vertexBuffer2
    )
);
```

## Step 2: Change a dynamic resource

If you specified a resource as dynamic and writable when you created it, you can later make changes to that resource.

[To change data in a dynamic resource](#)

1. Set up the new data for the resource. Declare a [D3D11\\_MAPPED\\_SUBRESOURCE](#) type variable and initialize it to zero. You'll use this variable to change the resource.
2. Disable graphics processing unit (GPU) access to the data that you want to change and get a pointer to the memory that contains the data. To get this pointer, pass [D3D11\\_MAP\\_WRITE\\_DISCARD](#) to the *MapType* parameter when you call [ID3D11DeviceContext::Map](#). Set this pointer to the address of the [D3D11\\_MAPPED\\_SUBRESOURCE](#) variable from the previous step.
3. Write the new data to the memory.
4. Call [ID3D11DeviceContext::Unmap](#) to reenable GPU access to the data when you're finished writing the new data.

Here is an example of changing a dynamic vertex buffer:

C++

```
// This might get called as the result of a click event to double the size
// of the triangle.
void TriangleRenderer::MapDoubleVertices()
{
    D3D11_MAPPED_SUBRESOURCE mappedResource;
    ZeroMemory(&mappedResource, sizeof(D3D11_MAPPED_SUBRESOURCE));
    float2 vertices[] =
    {
        float2(-1.0f, -1.0f),
        float2(0.0f, 1.0f),
        float2(1.0f, -1.0f),
    };
    // Disable GPU access to the vertex buffer data.
    auto m_d3dContext = m_deviceResources->GetD3DDeviceContext();
    m_d3dContext->Map(vertexBuffer2.Get(), 0, D3D11_MAP_WRITE_DISCARD, 0,
&mappedResource);
    // Update the vertex buffer here.
    memcpy(mappedResource.pData, vertices, sizeof(vertices));
    // Reenable GPU access to the vertex buffer data.
    m_d3dContext->Unmap(vertexBuffer2.Get(), 0);
}
```

## Remarks

### Using dynamic textures

We recommend that you create only one dynamic texture per format and possibly per size. We don't recommend creating dynamic mipmaps, cubes, and volumes because of the additional overhead in mapping every level. For mipmaps, you can specify [D3D11\\_MAP\\_WRITE\\_DISCARD](#) only on the top level. All levels are discarded by mapping

just the top level. This behavior is the same for volumes and cubes. For cubes, the top level and face 0 are mapped.

## Using dynamic buffers

When you call [Map](#) on a static vertex buffer while the GPU is using the buffer, you get a significant performance penalty. In this situation, [Map](#) must wait until the GPU is finished reading vertex or index data from the buffer before [Map](#) can return to the calling app, which causes a significant delay. Calling [Map](#) and then rendering from a static buffer several times per frame also prevents the GPU from buffering rendering commands because it must finish commands before returning the map pointer. Without buffered commands, the GPU remains idle until after the app is finished filling the vertex buffer or index buffer and issues a rendering command.

Ideally the vertex or index data would never change, but this is not always possible. There are many situations where the app needs to change vertex or index data every frame, perhaps even multiple times per frame. For these situations, we recommend that you create the vertex or index buffer with [D3D11\\_USAGE\\_DYNAMIC](#). This usage flag causes the runtime to optimize for frequent map operations. [D3D11\\_USAGE\\_DYNAMIC](#) is only useful when the buffer is mapped frequently; if data is to remain constant, place that data in a static vertex or index buffer.

To receive a performance improvement when you use dynamic vertex buffers, your app must call [Map](#) with the appropriate [D3D11\\_MAP](#) values. [D3D11\\_MAP\\_WRITE\\_DISCARD](#) indicates that the app doesn't need to keep the old vertex or index data in the buffer. If the GPU is still using the buffer when you call [Map](#) with [D3D11\\_MAP\\_WRITE\\_DISCARD](#), the runtime returns a pointer to a new region of memory instead of the old buffer data. This allows the GPU to continue using the old data while the app places data in the new buffer. No additional memory management is required in the app; the old buffer is reused or destroyed automatically when the GPU is finished with it.

### Note

When you map a buffer with [D3D11\\_MAP\\_WRITE\\_DISCARD](#), the runtime always discards the entire buffer. You can't preserve info in unmapped areas of the buffer by specifying a nonzero offset or limited size field.

There are cases where the amount of data the app needs to store per map is small, such as adding four vertices to render a sprite. [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#)

indicates that the app will not overwrite data already in use in the dynamic buffer. The [Map](#) call will return a pointer to the old data, which will allow the app to add new data in unused regions of the vertex or index buffer. The app must not modify vertices or indices that are used in a draw operation as they might still be in use by the GPU. We recommend that the app then use [D3D11\\_MAP\\_WRITE\\_DISCARD](#) after the dynamic buffer is full to receive a new region of memory, which discards the old vertex or index data after the GPU is finished.

The asynchronous query mechanism is useful to determine if vertices are still in use by the GPU. Issue a query of type [D3D11\\_QUERY\\_EVENT](#) after the last draw call that uses the vertices. The vertices are no longer in use when [ID3D11DeviceContext::GetData](#) returns S\_OK. When you map a buffer with [D3D11\\_MAP\\_WRITE\\_DISCARD](#) or no map values, you are always guaranteed that the vertices are synchronized properly with the GPU. But, when you map without map values, you will incur the performance penalty described earlier.

#### Note

The Direct3D 11.1 runtime, which is available starting with Windows 8, enables mapping dynamic constant buffers and shader resource views (SRVs) of dynamic buffers with [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#). The Direct3D 11 and earlier runtimes limited no-overwrite partial-update mapping to vertex or index buffers. To determine if a Direct3D device supports these features, call [ID3D11Device::CheckFeatureSupport](#) with [D3D11\\_FEATURE\\_D3D11\\_OPTIONS](#). [CheckFeatureSupport](#) fills members of a [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) structure with the device's features. The relevant members here are [MapNoOverwriteOnDynamicConstantBuffer](#) and [MapNoOverwriteOnDynamicBufferSRV](#).

## Related topics

[How to Use Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# How To: Create a Compute Shader

Article • 08/25/2021

A compute shader is an Microsoft High Level Shader Language (HLSL) programmable shader that uses generalized input and output memory access to support virtually any type of calculation. This topic shows how to create a compute shader. The compute shader technology is also known as the DirectCompute technology.

**To create a compute shader:**

1. Compile the HLSL shader code by calling [D3DCompileFromFile](#).

```
UINT flags = D3DCOMPILE_ENABLE_STRICTNESS;
#if defined( DEBUG ) || defined( _DEBUG )
    flags |= D3DCOMPILE_DEBUG;
#endif
    // Prefer higher CS shader profile when possible as CS 5.0 provides
    // better performance on 11-class hardware.
    LPCSTR profile = ( device->GetFeatureLevel() >=
D3D_FEATURE_LEVEL_11_0 ) ? "cs_5_0" : "cs_4_0";
    const D3D_SHADER_MACRO defines[] =
{
    "EXAMPLE_DEFINE", "1",
    NULL, NULL
};
    ID3DBlob* shaderBlob = nullptr;
    ID3DBlob* errorBlob = nullptr;
    HRESULT hr = D3DCompileFromFile( srcFile, defines,
D3D_COMPILE_STANDARD_FILE_INCLUDE,
                                entryPoint, profile,
                                flags, 0, &shaderBlob, &errorBlob
);
```

2. Create a compute shader using [ID3D11Device::CreateComputeShader](#).

```
ID3D11ComputeShader* g_pFinalPassCS = NULL;
pd3dDevice->CreateComputeShader( pBlobFinalPassCS->GetBufferPointer(),
                                    pBlobFinalPassCS-
>GetBufferSize(),
                                    NULL, &g_pFinalPassCS );
```

The following code example shows how to compile and create a compute shader.

## (!) Note

For this example code, you need the Windows SDK 8.0 and the d3dcompiler\_44.dll file from the %PROGRAM\_FILE%\Windows Kits\8.0\Redist\DXGI\<arch> folder in your path.

C++

```
#define _WIN32_WINNT 0x600
#include <stdio.h>

#include <d3d11.h>
#include <d3dcompiler.h>

#pragma comment(lib,"d3d11.lib")
#pragma comment(lib,"d3dcompiler.lib")

HRESULT CompileComputeShader( _In_ LPCWSTR srcFile, _In_ LPCSTR entryPoint,
                             _In_ ID3D11Device* device, _Outptr_ ID3DBlob** blob )
{
    if ( !srcFile || !entryPoint || !device || !blob )
        return E_INVALIDARG;

    *blob = nullptr;

    UINT flags = D3DCOMPILE_ENABLE_STRICTNESS;
#if defined( DEBUG ) || defined( _DEBUG )
    flags |= D3DCOMPILE_DEBUG;
#endif

    // We generally prefer to use the higher CS shader profile when possible
    // as CS 5.0 is better performance on 11-class hardware
    LPCSTR profile = ( device->GetFeatureLevel() >= D3D_FEATURE_LEVEL_11_0 )
        ? "cs_5_0" : "cs_4_0";

    const D3D_SHADER_MACRO defines[] =
    {
        "EXAMPLE_DEFINE", "1",
        NULL, NULL
    };

    ID3DBlob* shaderBlob = nullptr;
    ID3DBlob* errorBlob = nullptr;
    HRESULT hr = D3DCompileFromFile( srcFile, defines,
                                     D3D_COMPILE_STANDARD_FILE_INCLUDE,
                                         entryPoint, profile,
                                         flags, 0, &shaderBlob, &errorBlob );
    if ( FAILED(hr) )
```

```

    {
        if ( errorBlob )
        {
            OutputDebugStringA( (char*)errorBlob->GetBufferPointer() );
            errorBlob->Release();
        }

        if ( shaderBlob )
            shaderBlob->Release();

        return hr;
    }

    *blob = shaderBlob;

    return hr;
}

int main()
{
    // Create Device
    const D3D_FEATURE_LEVEL lvl[] = { D3D_FEATURE_LEVEL_11_1,
D3D_FEATURE_LEVEL_11_0,
                                            D3D_FEATURE_LEVEL_10_1,
D3D_FEATURE_LEVEL_10_0 };

    UINT createDeviceFlags = 0;
#ifdef _DEBUG
    createDeviceFlags |= D3D11_CREATE_DEVICE_DEBUG;
#endif

    ID3D11Device* device = nullptr;
    HRESULT hr = D3D11CreateDevice( nullptr, D3D_DRIVER_TYPE_HARDWARE,
nullptr, createDeviceFlags, lvl, _countof(lvl),
                                            D3D11_SDK_VERSION, &device, nullptr,
nullptr );
    if ( hr == E_INVALIDARG )
    {
        // DirectX 11.0 Runtime doesn't recognize D3D_FEATURE_LEVEL_11_1 as
a valid value
        hr = D3D11CreateDevice( nullptr, D3D_DRIVER_TYPE_HARDWARE, nullptr,
0, &lvl[1], _countof(lvl) - 1,
                                            D3D11_SDK_VERSION, &device, nullptr, nullptr
);
    }

    if ( FAILED(hr) )
    {
        printf("Failed creating Direct3D 11 device %08X\n", hr );
        return -1;
    }

    // Verify compute shader is supported
    if ( device->GetFeatureLevel() < D3D_FEATURE_LEVEL_11_0 )
    {

```

```

        D3D11_FEATURE_DATA_D3D10_X_HARDWARE_OPTIONS hwopts = { 0 } ;
        (void)device->CheckFeatureSupport(
            D3D11_FEATURE_D3D10_X_HARDWARE_OPTIONS, &hwopts, sizeof(hwopts) );
        if (
            !hwopts.ComputeShaders_Plus_RawAndStructuredBuffers_Via_Shader_4_x )
        {
            device->Release();
            printf( "DirectCompute is not supported by this device\n" );
            return -1;
        }
    }

    // Compile shader
    ID3DBlob *csBlob = nullptr;
    hr = CompileComputeShader( L"ExampleCompute.hlsl", "CSMain", device,
    &csBlob );
    if ( FAILED(hr) )
    {
        device->Release();
        printf("Failed compiling shader %08X\n", hr );
        return -1;
    }

    // Create shader
    ID3D11ComputeShader* computeShader = nullptr;
    hr = device->CreateComputeShader( csBlob->GetBufferPointer(), csBlob-
    >GetBufferSize(), nullptr, &computeShader );

    csBlob->Release();

    if ( FAILED(hr) )
    {
        device->Release();
    }

    printf("Success\n");

    // Clean up
    computeShader->Release();

    device->Release();

    return 0;
}

```

The preceding code example compiles the compute shader code in the ExampleCompute.hlsl file. Here is the code in ExampleCompute.hlsl:

hlsl

```
//-----
```

```

// File: BasicCompute11.hlsl
//
// This file contains the Compute Shader to perform array A + array B
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//-----
-----

#ifndef USE_STRUCTURED_BUFFERS

struct BufType
{
    int i;
    float f;
#ifdef TEST_DOUBLE
    double d;
#endif
};

StructuredBuffer<BufType> Buffer0 : register(t0);
StructuredBuffer<BufType> Buffer1 : register(t1);
RWStructuredBuffer<BufType> BufferOut : register(u0);

[numthreads(1, 1, 1)]
void CSMain( uint3 DTid : SV_DispatchThreadID )
{
    BufferOut[DTid.x].i = Buffer0[DTid.x].i + Buffer1[DTid.x].i;
    BufferOut[DTid.x].f = Buffer0[DTid.x].f + Buffer1[DTid.x].f;
#ifdef TEST_DOUBLE
    BufferOut[DTid.x].d = Buffer0[DTid.x].d + Buffer1[DTid.x].d;
#endif
}

#else // The following code is for raw buffers

ByteAddressBuffer Buffer0 : register(t0);
ByteAddressBuffer Buffer1 : register(t1);
RWByteAddressBuffer BufferOut : register(u0);

[numthreads(1, 1, 1)]
void CSMain( uint3 DTid : SV_DispatchThreadID )
{
#ifdef TEST_DOUBLE
    int i0 = asint( Buffer0.Load( DTid.x*16 ) );
    float f0 = asfloat( Buffer0.Load( DTid.x*16+4 ) );
    double d0 = asdouble( Buffer0.Load( DTid.x*16+8 ), Buffer0.Load(
DTid.x*16+12 ) );
    int i1 = asint( Buffer1.Load( DTid.x*16 ) );
    float f1 = asfloat( Buffer1.Load( DTid.x*16+4 ) );
    double d1 = asdouble( Buffer1.Load( DTid.x*16+8 ), Buffer1.Load(
DTid.x*16+12 ) );

    BufferOut.Store( DTid.x*16, asuint(i0 + i1) );
    BufferOut.Store( DTid.x*16+4, asuint(f0 + f1) );

```

```
    uint d1, dh;
    asuint( d0 + d1, dl, dh );

    BufferOut.Store( DTid.x*16+8, dl );
    BufferOut.Store( DTid.x*16+12, dh );
#else
    int i0 = asint( Buffer0.Load( DTid.x*8 ) );
    float f0 = asfloat( Buffer0.Load( DTid.x*8+4 ) );
    int i1 = asint( Buffer1.Load( DTid.x*8 ) );
    float f1 = asfloat( Buffer1.Load( DTid.x*8+4 ) );

    BufferOut.Store( DTid.x*8, asuint(i0 + i1) );
    BufferOut.Store( DTid.x*8+4, asuint(f0 + f1) );
#endif // TEST_DOUBLE
}

#endif // USE_STRUCTURED_BUFFERS
```

## Related topics

- [Compute Shader Overview](#)
- [How to Use Direct3D 11](#)
- [BasicCompute11 sample application ↗](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Design a Hull Shader

Article • 08/19/2020

A hull shader is the first of three stages that work together to implement [tessellation](#) (the other two stages are the tessellator and a domain shader). This topics shows how to design a hull shader.

A hull shader requires two functions, the main hull shader and a patch constant function. The hull shader implements calculations on each control point; the hull shader also calls the patch constant function which implements calculations on each patch.

After you have designed a hull shader, see [How To: Create a Hull Shader](#) to learn how to create a hull shader.

## To design a hull shader

1. Define hull shader input control and output control points.

```
// Input control point
struct VS_CONTROL_POINT_OUTPUT
{
    float3 vPosition : WORLDPOS;
    float2 vUV       : TEXCOORD0;
    float3 vTangent  : TANGENT;
};

// Output control point
struct BEZIER_CONTROL_POINT
{
    float3 vPosition   : BEZIERPOS;
};
```

2. Define output patch constant data.

```
// Output patch constant data.
struct HS_CONSTANT_DATA_OUTPUT
{
    float Edges[4]      : SV_TessFactor;
    float Inside[2]     : SV_InsideTessFactor;

    float3 vTangent[4]  : TANGENT;
    float2 vUV[4]       : TEXCOORD;
    float3 vTanUCorner[4] : TANUCORNER;
    float3 vTanVCorner[4] : TANVCORNER;
```

```
    float4 vCWts : TANWEIGHTS;  
};
```

For a quad domain, [SV\\_TessFactor](#) defines 4 edge tessellation factors (to tessellate the edges), since the fixed function tessellator needs to know how much to tessellate. The required outputs are different for the triangle and isoline domains.

The fixed function tessellator doesn't look at any other hull shader outputs, such as other patch constant data or any of the control points. The domain shader -- which is invoked for each point the fixed function tessellator generates -- will see as its input all the hull shader's output control points and all the output patch constant data; the shader evaluates the patch at its location.

3. Define a patch constant function. A patch constant function executes once for each patch to calculate any data that is constant for the entire patch (as opposed to per-control point data, which is computed in the hull shader).

```
#define MAX_POINTS 32  
  
// Patch Constant Function  
HS_CONSTANT_DATA_OUTPUT SubDToBezierConstantsHS(  
    InputPatch<VS_CONTROL_POINT_OUTPUT, MAX_POINTS> ip,  
    uint PatchID : SV_PrimitiveID )  
{  
    HS_CONSTANT_DATA_OUTPUT Output;  
  
    // Insert code to compute Output here  
  
    return Output;  
}
```

Properties of the patch constant function include:

- One input specifies a variable containing a patch id, and is identified by the [SV\\_PrimitiveID](#) system value (see [semantics](#) in shader model 4).
- One input parameter is the input control points, declared in [VS\\_CONTROL\\_POINT\\_OUTPUT](#) in this example. A patch function can see all the input control points for each patch, there are 32 control points per patch in this example.
- As a minimum, the function must calculate per-patch tessellation factors for the tessellator stage which are identified with [SV\\_TessFactor](#). A quad domain requires four tessellation factors for the edges and two additional factors (identified by [SV\\_InsideTessFactor](#)) for tessellating the inside of the patch. The

- fixed function tessellator doesn't look at any other hull shader outputs (such as the patch constant data or any of the control points).
- The outputs are usually defined by a structure and is identified by **HS\_CONSTANT\_DATA\_OUTPUT** in this example; the structure depends on the domain type and would be different for triangle or isoline domains.

A domain shader on the other hand is invoked for each point the fixed function tessellator generates and needs to see the output control points and the output patch constant data (both from the hull shader) to evaluate a patch at its location.

- Define a hull shader. A hull shader identifies the properties of a patch including a patch constant function. A hull shader is invoked once for each output control point.

```
[domain("quad")]
[partitioning("integer")]
[outputtopology("triangle_cw")]
[outputcontrolpoints(16)]
[patchconstantfunc("SubDToBezierConstantsHS")]
BEZIER_CONTROL_POINT SubDToBezierHS(
    InputPatch<VS_CONTROL_POINT_OUTPUT, MAX_POINTS> ip,
    uint i : SV_OutputControlPointID,
    uint PatchID : SV_PrimitiveID )
{
    VS_CONTROL_POINT_OUTPUT Output;

    // Insert code to compute Output here.

    return Output;
}
```

A hull shader uses the following attributes:

- A **domain** attribute.
- A **partitioning** attribute.
- An **outputtopology** attribute.
- An **outputcontrolpoints** attribute.
- A **patchconstantfunc** attribute. A hull shader calculates output control points, there are 16 output Bezier control points in this example.

All the input control points (identified by **VS\_CONTROL\_POINT\_OUTPUT**) are visible to each hull shader invocation. In this example, there are 32 input control points.

A hull shader is invoked once per output control point (identified with **SV\_OutputControlPointID**) for each patch (identified with **SV\_PrimitiveID**). The purpose

of this particular shader is to calculate output  $i$ , which was defined to be a BEZIER control point (this example has 16 output control points defined by `outputcontrolpoints`).

A hull shader runs a routine once per patch (the patch constant function) to compute patch-constant data (tessellation factors as a minimum). Separately, a hull shader runs a patch constant function (called `SubDToBezierConstantsHS`) on each patch to compute patch-constant data such as tessellation factors for the tessellator stage.

## Related topics

[How to Use Direct3D 11](#)

[Tessellation Overview](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Create a Hull Shader

Article • 08/23/2019

A hull shader is the first of three stages that work together to implement [tessellation](#). The hull-shader outputs drive the tessellator stage, as well as the domain-shader stage. This topic shows how to create a hull shader.

A hull shader transforms a set of input control points (from a vertex shader) into a set of output control points. The number of input and output points can vary in contents and number depending on the transform (a typical transformation would be a basis transformation).

A hull shader also outputs patch constant information, such as tessellation factors, for a domain shader and the tessellator. The fixed-function tessellator stage uses the tessellation factors as well as other state declared in a hull shader to determine how much to tessellate.

## To create a hull shader

1. Design a hull shader. See [How To: Design a Hull Shader](#).
2. Compile the shader code
3. Create a hull-shader object using [ID3D11Device::CreateHullShader](#).

```
HRESULT CreateHullShader(  
    const void *pShaderBytecode,  
    SIZE_T BytecodeLength,  
    ID3D11ClassLinkage *pClassLinkage,  
    ID3D11HullShader **ppHullShader  
)
```

4. Initialize the pipeline stage using [ID3D11DeviceContext::HSSetShader](#).

```
void HSSetShader(  
    ID3D11HullShader *pHullShader,  
    ID3D11ClassInstance *const *ppClassInstances,  
    UINT NumClassInstances  
)
```

A domain shader must be bound to the pipeline if a hull shader is bound. In particular, it is not valid to directly stream out hull shader control points with the geometry shader.

## Related topics

[How to Use Direct3D 11](#)

[Tessellation Overview](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Initialize the Tessellator Stage

Article • 08/23/2019

In general, tessellation expands the compact, user-defined, model of a patch into geometry that contains a programmable amount of detail. The geometry is typically a set of triangles that represents detailed surface geometry. This topic shows how to initialize the tessellator stage.

The tessellator stage is the second of three stages that work together to tessellate or tile a surface. The first stage is the hull-shader stage; it operates once per patch and configures how the next stage (the fixed function tessellator) behaves. A hull shader also generates user-defined outputs such as output-control points and patch constants that are sent past the tessellator directly to the third stage, the domain-shader stage. A domain shader is invoked once per tessellator-stage point and evaluates surface positions.

The tessellator stage is a fixed function stage, there is no shader to generate, and no state to set. It receives all of its setup state from the hull-shader stage; once the hull-shader stage has been initialized, the tessellator stage is automatically initialized.

## To initialize the tessellator stage

- Initialize the hull-shader stage using [ID3D11DeviceContext::HSSetShader](#).

```
void HSSetShader(
    ID3D11HullShader *pHullShader,
    ID3D11ClassInstance *const *ppClassInstances,
    UINT NumClassInstances
);
```

*ppClassInstances* is a pointer to an array of shader interfaces, represented by [ID3D11ClassInstance](#) pointers and the number of interfaces, represented by *NumClassInstances*. If not used, these parameters can be set to **NULL** and 0 respectively.

After the hull-shader stage is initialized, you should also initialize the domain-shader stage.

## Related topics

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# How To: Design a Domain Shader

Article • 08/19/2020

A domain shader is the third of three stages that work together to implement [tessellation](#). The domain shader generates the surface geometry from the transformed control points from a hull shader and the UV coordinates. This topics shows how to design a domain shader.

A domain shader is invoked once for each point generated by the fixed function tessellator. The inputs are the UV[W] coordinates of the point on the patch, as well as all of the output data from the hull shader including control points and patch constants. The output is a vertex defined in whatever way is desired. If the output is being sent to the pixel shader, the output must include a position (denoted with a SV\_Position semantic).

## To design a domain shader

1. Define the domain attribute.

```
[domain("quad")]
```

The domain is defined for a quad patch.

2. Declare the location on the hull with the [domain location](#) system value.

- For a quad patch, use a float2.
- For a tri patch, use a float3 (for barycentric coordinates)
- For an isoline, use a float2.

Therefore, the domain location for a quad patch looks like this:

```
float2 UV : SV_DomainLocation
```

3. Define the other inputs.

The other inputs come from the hull shader and are user defined. This includes the input control points for patch, of which there can be between 1 and 32 points, and input patch constant data.

The control points are user defined, usually with a structure such as this one (defined in [How To: Design a Hull Shader](#)):

```
const OutputPatch<BEZIER_CONTROL_POINT, 16> bezpatch
```

The patch constant data is also user defined, and might look like this one (defined in [How To: Design a Hull Shader](#)):

```
HS_CONSTANT_DATA_OUTPUT input
```

4. Add user-defined code to compute the outputs; this makes up the body of the domain shader.

This structure contains user-defined domain shader outputs.

```
struct DS_OUTPUT
{
    float3 vNormal      : NORMAL;
    float2 vUV          : TEXCOORD;
    float3 vTangent     : TANGENT;
    float3 vBiTangent   : BITANGENT;

    float4 vPosition    : SV_POSITION;
};
```

The function takes each input UV (from the tessellator) and evaluates the Bezier patch at this position.

```
[domain("quad")]
DS_OUTPUT BezierEvalDS( HS_CONSTANT_DATA_OUTPUT input,
                        float2 UV : SV_DomainLocation,
                        const OutputPatch<BEZIER_CONTROL_POINT, 16>
bezpatch )
{
    DS_OUTPUT Output;

    // Insert code to compute the output here.

    return Output;
}
```

The function is invoked once for each point generated by the fixed function tessellator. Since this example uses a quad patch, the input domain location ([SV\\_DomainLocation](#)) is a float2 (UV); a tri patch would have a float3 input location (UVW barycentric coordinates), and an isoline would have a float2 input domain location.

The other inputs for the function come from the hull shader directly. In this example it is 16 control points each one being a **BEZIER\_CONTROL\_POINT**, as well as patch constant data ([HS\\_CONSTANT\\_DATA\\_OUTPUT](#)). The output is a vertex containing any data desired - **DS\_OUTPUT** in this example.

After designing a domain shader, see [How To: Create a Domain Shader](#).

## Related topics

[How to Use Direct3D 11](#)

[Tessellation Overview](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Create a Domain Shader

Article • 08/23/2019

A domain shader is the third of three stages that work together to implement [tessellation](#). The inputs for the domain-shader stage come from a hull shader. This topic shows how to create a domain shader.

A domain shader transforms surface geometry (created by the fixed-function tessellator stage) using hull shader output-control points, hull shader output patch-constant data, and a single set of tessellator uv coordinates.

## To create a domain shader

1. Design a domain shader. See [How To: Design a Domain Shader](#).
2. Compile the shader code.
3. Create a domain-shader object using [ID3D11Device::CreateDomainShader](#).

```
HRESULT CreateDomainShader(
    const void *pShaderBytecode, // 
    SIZE_T BytecodeLength, // 
    ID3D11ClassLinkage *pClassLinkage, // 
    ID3D11DomainShader **ppDomainShader
);
```

4. Initialize the pipeline stage using [ID3D11DeviceContext::DSSetShader](#).

```
void DSSetShader(
    ID3D11DomainShader *pDomainShader, // 
    ID3D11ClassInstance *const *ppClassInstances,
    UINT NumClassInstances
);
```

A domain shader must be bound to the pipeline if a hull shader is bound. In particular, it is not valid to directly stream out hull shader control points with the geometry shader.

## Related topics

[How to Use Direct3D 11](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# How To: Compile a Shader

Article • 08/25/2021

You typically use the [fxc.exe](#) HLSL code compiler as part of the build process to compile shader code. For more info about this, see [Compiling Shaders](#). This topic shows how to use the [D3DCompileFromFile](#) function at run time to compile shader code.

To compile a shader:

- Compile HLSL shader code by calling [D3DCompileFromFile](#).

C++

```
UINT flags = D3DCOMPILE_ENABLE_STRICTNESS;
#if defined( DEBUG ) || defined( _DEBUG )
    flags |= D3DCOMPILE_DEBUG;
#endif
    // Prefer higher CS shader profile when possible as CS 5.0 provides
    // better performance on 11-class hardware.
    LPCSTR profile = ( device->GetFeatureLevel() >=
D3D_FEATURE_LEVEL_11_0 ) ? "cs_5_0" : "cs_4_0";
    const D3D_SHADER_MACRO defines[] =
    {
        "EXAMPLE_DEFINE", "1",
        NULL, NULL
    };
    ID3DBlob* shaderBlob = nullptr;
    ID3DBlob* errorBlob = nullptr;
    HRESULT hr = D3DCompileFromFile( srcFile, defines,
D3D_COMPILE_STANDARD_FILE_INCLUDE,
                                entryPoint, profile,
                                flags, 0, &shaderBlob, &errorBlob
    );
```

The following code example shows how to compile various shaders.

## ⓘ Note

For this example code, you need the Windows SDK 8.0 and the d3dcompiler\_44.dll file from the %PROGRAM\_FILE%\Windows Kits\8.0\Redist\DXGI\<arch> folder in your path. Windows Store apps support run time compilation for development but not for deployment.

C++

```
#define _WIN32_WINNT 0x600
#include <stdio.h>

#include <d3dcompiler.h>

#pragma comment(lib,"d3dcompiler.lib")

HRESULT CompileShader( _In_ LPCWSTR srcFile, _In_ LPCSTR entryPoint, _In_
LPCSTR profile, _Outptr_ ID3DBlob** blob )
{
    if ( !srcFile || !entryPoint || !profile || !blob )
        return E_INVALIDARG;

    *blob = nullptr;

    UINT flags = D3DCOMPILE_ENABLE_STRICTNESS;
#ifndef DEBUG || defined( _DEBUG )
    flags |= D3DCOMPILE_DEBUG;
#endif

    const D3D_SHADER_MACRO defines[] =
    {
        "EXAMPLE_DEFINE", "1",
        NULL, NULL
    };

    ID3DBlob* shaderBlob = nullptr;
    ID3DBlob* errorBlob = nullptr;
    HRESULT hr = D3DCompileFromFile( srcFile, defines,
D3D_COMPILE_STANDARD_FILE_INCLUDE,
                                entryPoint, profile,
                                flags, 0, &shaderBlob, &errorBlob );
    if ( FAILED(hr) )
    {
        if ( errorBlob )
        {
            OutputDebugStringA( (char*)errorBlob->GetBufferPointer() );
            errorBlob->Release();
        }

        if ( shaderBlob )
            shaderBlob->Release();
    }

    return hr;
}

*blob = shaderBlob;

return hr;
}

int main()
{
    // Compile vertex shader shader
```

```

ID3DBlob *vsBlob = nullptr;
HRESULT hr = CompileShader( L"BasicHLSL11_VS.hlsl", "VSMain",
"vs_4_0_level_9_1", &vsBlob );
if ( FAILED(hr) )
{
    printf("Failed compiling vertex shader %08X\n", hr );
    return -1;
}

// Compile pixel shader shader
ID3DBlob *psBlob = nullptr;
hr = CompileShader( L"BasicHLSL11_PS.hlsl", "PSMain",
"ps_4_0_level_9_1", &psBlob );
if ( FAILED(hr) )
{
    vsBlob->Release();
    printf("Failed compiling pixel shader %08X\n", hr );
    return -1;
}

printf("Success\n");

// Clean up
vsBlob->Release();
psBlob->Release();

return 0;
}

```

The preceding code example compiles the pixel and vertex shader code blocks in the BasicHLSL11\_PS.hlsl and BasicHLSL11\_VS.hlsl files. Here is the code in BasicHLSL11\_PS.hlsl:

```

hlsl

//-----
-----
// File: BasicHLSL11_PS.hlsl
//
// The pixel shader file for the BasicHLSL11 sample.
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//-----
-----

//-----
-----
// Globals
//-----
-----

cbuffer cbPerObject : register( b0 )

```

```

{
    float4      g_vObjectColor           : packoffset( c0 );
};

cbuffer cbPerFrame : register( b1 )
{
    float3      g_vLightDir            : packoffset( c0 );
    float       g_fAmbient             : packoffset( c0.w );
};

//-----
//-----
// Textures and Samplers
//-----

Texture2D   g_txDiffuse : register( t0 );
SamplerState g_samLinear : register( s0 );

//-----
//-----
// Input / Output structures
//-----

struct PS_INPUT
{
    float3 vNormal      : NORMAL;
    float2 vTexcoord    : TEXCOORD0;
};

//-----
//-----
// Pixel Shader
//-----

float4 PSMain( PS_INPUT Input ) : SV_TARGET
{
    float4 vDiffuse = g_txDiffuse.Sample( g_samLinear, Input.vTexcoord );

    float fLighting = saturate( dot( g_vLightDir, Input.vNormal ) );
    fLighting = max( fLighting, g_fAmbient );

    return vDiffuse * fLighting;
}

```

Here is the code in BasicHLSL11\_VS.hlsl:

```

hlsl

//-----
//-----
// File: BasicHLSL11_VS.hlsl
//
// The vertex shader file for the BasicHLSL11 sample.

```

```

// Copyright (c) Microsoft Corporation. All rights reserved.
// -----
// Global

cbuffer cbPerObject : register( b0 )
{
    matrix      g_mWorldViewProjection   : packoffset( c0 );
    matrix      g_mWorld                 : packoffset( c4 );
};

// Input / Output structures
// -----
struct VS_INPUT
{
    float4 vPosition     : POSITION;
    float3 vNormal       : NORMAL;
    float2 vTexcoord     : TEXCOORD0;
};

struct VS_OUTPUT
{
    float3 vNormal       : NORMAL;
    float2 vTexcoord     : TEXCOORD0;
    float4 vPosition     : SV_POSITION;
};

// Vertex Shader
// -----
VS_OUTPUT VSMain( VS_INPUT Input )
{
    VS_OUTPUT Output;

    Output.vPosition = mul( Input.vPosition, g_mWorldViewProjection );
    Output.vNormal = mul( Input.vNormal, (float3x3)g_mWorld );
    Output.vTexcoord = Input.vTexcoord;

    return Output;
}

```

## Related topics

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How to: Record a Command List

Article • 08/23/2019

A [command list](#) is a recorded list of rendering commands. This topic shows how to create and record a [command list](#). Use a command list to record render commands and play them back later. A command list is convenient for splitting rendering tasks between threads.

## To record a command list

1. A command list must be created from a deferred context, which contains device state and rendering actions. Given a device, create a deferred context by calling [ID3D11Device::CreateDeferredContext](#).

```
HRESULT hr;
ID3D11DeviceContext* pDeferredContext = NULL;

hr = g_pd3dDevice->CreateDeferredContext(0, &pDeferredContext);
```

2. Use the deferred context to render.

```
float ClearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
pDeferredContext->ClearRenderTargetView( g_pRenderTargetView,
ClearColor );

// Add additional rendering commands
...
```

This simple example clears a render target, but you could add additional render commands.

3. Create/record a command list by calling [ID3D11DeviceContext::FinishCommandList](#) and passing a pointer to an uninitialized [ID3D11CommandList](#) interface.

```
ID3D11CommandList* pd3dCommandList = NULL;
HRESULT hr;
hr = pDeferredContext->FinishCommandList(FALSE, &pd3dCommandList);
```

When the method returns, a command list is created containing all the render commands and an interface is returned to the application.

The boolean parameter tells the runtime what to do with the pipeline state in the deferred context. **TRUE** means restore the state of the device context to its pre-command list condition when recording is completed, **FALSE** means do not change the state after recording. This means that the device context will reflect the state changes contained in the command list.

To see an example for playing back a command list, see [How to: Play Back a Command List](#).

## Related topics

[Command List](#)

[How to Use Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How to: Play Back a Command List

Article • 08/23/2019

A [command list](#) is a recorded list of rendering commands. Use a command list to pre-record drawing commands and play them back later. This topic shows how to play back a [command list](#). A command list can be used to split rendering tasks between threads.

This section describes how to play back a command list. For recording a command list, see [How to: Record a Command List](#).

## To play back a command list

- Call [ID3D11DeviceContext::ExecuteCommandList](#) and pass a valid [ID3D11CommandList](#) object.

```
if(g_pd3dCommandList)
{
    g_pImmediateContext->ExecuteCommandList(g_pd3dCommandList, TRUE);
```

[ExecuteCommandList](#) must be executed on the [immediate context](#) for recorded commands to be run on the graphics processing unit (GPU). Use the immediate context to feed commands to the GPU for execution, use a deferred context to record commands for playback onto another command list. When you call [ExecuteCommandList](#) onto another deferred context, you create a ‘merged’ deferred command list. To run the commands on the merged deferred command list, you need to execute them on the immediate context.

## Related topics

[Command List](#)

[How to Use Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# How To: Check for Driver Support

Article • 08/23/2019

This topic shows how to determine whether multithreading features (including [resource creation](#) and [command lists](#)) are supported for hardware acceleration.

We recommend for applications to check for graphics hardware support of multithreading. If the driver and graphics hardware do not support multithreaded object creation, performance can be limited in the following ways:

- Creating multiple objects (even of different types) at the same time might be limited.
- Creating an object while rendering graphics commands by using an [immediate context](#) might be limited. For example, if hardware does not support multithreading, an application should avoid creating on a background thread an object that requires a very long time to create. A create operation that takes very long can block immediate context rendering and increase the risk of a visual frame rate stutter.

The runtime supports multithreading and command lists regardless of driver and hardware support; if there is no driver and hardware support for either multithreads or command lists, the runtime will emulate the functionality. For more information about multithreading, see [Introduction to Multithreading in Direct3D 11](#).

To check for driver support for multithreading:

1. Initialize an [ID3D11Device](#) interface object. By default, multithreading is enabled.
2. Call [ID3D11Device::CheckFeatureSupport](#). Pass the [D3D11\\_FEATURE\\_THREADING](#) value to the *Feature* parameter, pass the [D3D11\\_FEATURE\\_DATA\\_THREADING](#) structure to the *pFeatureSupportData* parameter, and pass the size of the [D3D11\\_FEATURE\\_DATA\\_THREADING](#) structure to the *FeatureSupportDataSize* parameter.
3. If the [ID3D11Device::CheckFeatureSupport](#) method succeeds, the [D3D11\\_FEATURE\\_DATA\\_THREADING](#) structure that you passed in the previous step will be initialized with information about multithreading support.
  - If [DriverConcurrentCreates](#) is **TRUE**, a driver can create more than one resource at the same time (concurrently) on different threads.If [DriverCommandLists](#) is **TRUE**, the driver supports command lists. That is, rendering commands issued by an immediate context can be concurrent with object creation on separate threads with low risk of a frame rate stutter.

- If `DriverConcurrentCreates` is `FALSE`, a driver does not support concurrent creation, which means the amount of concurrency possible is extremely limited. The graphics hardware cannot create objects of different types on different threads simultaneously. Additionally, the graphics hardware cannot use an immediate context to issue render commands while the graphics hardware attempts to create a resource on another thread.

## Related topics

[How to Use Direct3D 11](#)

[Multithreading](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# What's new in Direct3D 11

Article • 08/19/2020

This section describes features added in Direct3D 11, Direct3D 11.1, and Direct3D 11.2.

Microsoft Direct3D 11 is an extension of the Microsoft Direct3D 10/Microsoft Direct3D 10.1 rendering API. For more introductory material about using Direct3D 11, see the [Programming Guide for Direct3D 10](#).

## In this section

Topic	Description
Direct3D 11 Features	The programming guide contains information about how to use the Direct3D 11 programmable pipeline to create realtime 3D graphics for games, and for scientific and desktop applications.
Direct3D 11.1 Features	The following functionality has been added in Direct3D 11.1, which is included with Windows 8, Windows RT, and Windows Server 2012. Partial support for <a href="#">Direct3D 11.1</a> is available on Windows 7 and Windows Server 2008 R2 via the <a href="#">Platform Update for Windows 7</a> , which is available through the <a href="#">Platform Update for Windows 7</a> .
Direct3D 11.2 Features	The following functionality has been added in Direct3D 11.2, which is included with Windows 8.1, Windows RT 8.1, and Windows Server 2012 R2.
Direct3D 11.3 Features	The following sections describe the functionality has been added in Direct3D 11.3. These features are also available in Direct3D 12.
Direct3D 11.4 Features	The following functionality has been added in Direct3D 11.4.
Features Introduced In Previous Releases	Discover what new features have been added to the previous SDK updates:

## Related topics

[Direct3D 11 Graphics](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11 Features

Article • 10/20/2020

The programming guide contains information about how to use the Direct3D 11 programmable pipeline to create realtime 3D graphics for games, and for scientific and desktop applications.

- [Compute Shader](#)
- [Dynamic Shader Linking](#)
- [Multithreading](#)
- [Tessellation](#)
- [Full Listing of Features](#)
- [Features Added in Previous Releases](#)
- [Related topics](#)

## Compute Shader

A compute shader is a programmable shader designed for general-purpose data-parallel processing. In other words, compute shaders allow a GPU to be used as a general-purpose parallel processor. The compute shader is similar to the other programmable pipeline shaders (such as vertex, pixel, geometry) in the way that it accesses inputs and outputs. The compute shader technology is also known as the DirectCompute technology. A compute shader is integrated into Direct3D and is accessible through a Direct3D device. It can directly share memory resources with graphics shaders by using the Direct3D device. However, it is not directly connected to other shader stages.

A compute shader is designed for mass-market applications that perform computations at interactive rates, when the cost of transitioning between the API (and its associated software stack) and a CPU would consume too much overhead.

A compute shader has its own set of states. A compute shader does not necessarily have a forced 1-1 mapping to either input records (like a vertex shader does) or output records (like the pixel shader does). Some features of the graphics shader are supported, but others have been removed so that new compute shader-specific features could be added.

To support the compute shader-specific features, several new resource types are now available, such as read/write buffers, textures, and structured buffers.

See [Compute Shader Overview](#) for additional information.

# Dynamic Shader Linking

Rendering systems must deal with significant complexity when they manage shaders, while providing the opportunity to optimize shader code. This becomes an even greater challenge because shaders must support a variety of different materials in a rendered scene across various hardware configurations. To address this challenge, shader developers have often resorted to one of two general approaches. They have either created fully featured large, general-purpose shaders that can be used by a wide variety of scene items, which trade off some performance for flexibility, or created individual shaders for each geometry stream, material type, or light type combination needed.

These large, general-purpose shaders handle this challenge by recompiling the same shader with different preprocessor definitions, and the latter method uses brute-force developer power to achieve the same result. The shader permutation explosion has often been a problem for developers who must now manage thousands of different shader permutations within their game and asset pipeline.

Direct3D 11 and shader model 5 introduce object-oriented language constructs and provide runtime support of shader linking to help developers program shaders.

See [Dynamic Linking](#) for additional information.

# Multithreading

Many graphics applications are CPU-bound because of costly activities such as scene graph traversal, object sorting, and physics simulations. Because multicore systems are becoming increasingly available, Direct3D 11 has improved its multithreading support to enable efficient interaction between multiple CPU threads and the D3D11 graphics APIs.

Direct3D 11 enables the following functionality to support multithreading:

- Concurrent objects are now created in separate threads — Making entry-point functions that create objects free-threaded makes it possible for many threads to create objects simultaneously. For example, an application can now compile a shader or load a texture on one thread while rendering on another.
- Command lists can be created on multiple threads — A command list is a recorded sequence of graphics commands. With Direct3D 11, you can create command lists on multiple CPU threads, which enables parallel traversal of the scene database or physics processing on multiple threads. This frees the main rendering thread to dispatch command buffers to the hardware.

See [MultiThreading](#) for additional information.

# Tessellation

Tessellation can be used to render a single model with varying levels of detail. This approach generates a more geometrically accurate model that depends on the level of detail required for a scene. Use tessellation in a scene where the level of detail allows a lower geometry model, which reduces the demand on memory bandwidth consumed during rendering.

In Direct3D, tessellation is implemented on the GPU to calculate a smoother curved surface from a coarse (less detailed) input patch. Each (quad or triangle) patch face is subdivided into smaller triangular faces that better approximate the surface that you want.

For information about implementing tessellation in the graphics pipeline, see [Tessellation Overview](#).

## Full Listing of Features

This is a complete list of the features in Direct3D 11.

- You can run Direct3D 11 on [downlevel hardware](#) by specifying a [feature level](#) when you create a device.
- You can perform tessellation (see [Tessellation Overview](#)) by using the following shader types:
  - Hull Shader
  - Domain Shader
- Direct3D 11 supports multithreading (see [MultiThreading](#))
  - Multithread resource/shader/object creation
  - Multithreaded Display list creation
- Direct3D 11 expands shaders with the following features (see [Shader Model 5](#))
  - Addressable resources - textures, constant buffers, and samplers
  - Additional resource types, such as read/write buffers and textures (see [New Resource Types](#)).
  - Subroutines
  - Compute shader (see [Compute Shader Overview](#)) - A shader that speeds up computations by dividing the problem space between several software threads or groups of threads, and sharing data among shader registers to significantly

reduce the amount of data required to input into a shader. Algorithms that the compute shader can significantly improve include post processing, animation, physics, and artificial intelligence.

- Geometry shader (see [Geometry Shader Features](#))
  - Instancing - Allows the geometry shader to output a maximum of 1024 vertices, or any combination of instances and vertices up to 1024 (maximum of 32 instances of 32 vertices each).
- Pixel shader
  - Coverage as PS Input
  - Programmable Interpolation of Inputs - The pixel shader can evaluate attributes within the pixel, anywhere on the multisample grid
  - Centroid sampling of attributes must obey the following rules:
    - If all samples in the primitive are covered, the attribute is evaluated at the pixel center regardless of whether the sample pattern has a sample location at the pixel center.
    - Otherwise, the attribute is evaluated at the first covered sample, that is, the sample with the lowest index among all sample indexes. In this situation, sample coverage is determined after applying the logical AND operation to the coverage and the sample-mask rasterizer state.
    - If no samples are covered (such as on helper pixels that are executed off the bounds of a primitive to fill out 2x2 pixel stamps), the attribute is evaluated in one of the following ways:
      - If the sample-mask rasterizer state is a subset of the samples in the pixel, the first sample that is covered by the sample-mask rasterizer state is the evaluation point.
      - Otherwise, in the full sample-mask condition, the pixel center is the evaluation point.
- Direct3D 11 expands textures (see [Textures Overview](#)) with the following features
  - Gather4
    - Support for multi-component textures - specify a channel to load from
    - Support for programmable offsets
  - Streaming
    - Texture clamps to limit WDDM preload

- 16K texture limits
  - Require 8-bits of subtexel and sub-mip precision on texture filtering
  - New texture compression formats (1 new LDR format and 1 new HDR format)
- Direct3D 11 supports conservative oDepth - This algorithm allows a pixel shader to compare the per-pixel depth value of the pixel shader with that in the rasterizer. The result enables early depth culling operations while maintaining the ability to output oDepth from a pixel shader.
- Direct3D 11 supports large memory
  - Allow for resources > 4GB
  - Keep indices of resources 32bit, but resource larger
- Direct3D 11 supports stream output improvements
  - Addressable Stream output
  - Increase Stream output count to 4
  - Change all stream output buffers to be multi-element
- Direct3D 11 supports Shader Model 5 (see [Shader Model 5](#))
  - Doubles with denorms
  - Count bits set instruction
  - Find first bit set instruction
  - Carry/Overflow handling
  - Bit reversal instructions for FFTs
  - Conditional Swap intrinsic
  - Resinfo on buffers
  - Reduced-precision reciprocal
  - Shader conversion instructions - fp16 to fp32 and vice versa
  - Structured buffer, which is a new type of buffer containing structured elements.
- Direct3D 11 supports read-only depth or stencil views
  - Disables writes to the part that is read-only, allows for using texture as input and for depth-culling
- Direct3D 11 supports draw indirect - Direct3D 10 implements DrawAuto, which takes content (generated by the GPU) and renders it (on the GPU). Direct3D 11 generalizes DrawAuto so that it can be called by a Compute Shader using DrawInstanced and DrawIndexedInstanced.
- Direct3D 11 supports miscellaneous features
  - Floating-point viewports
  - Per-resource mipmap clamping

- Depth Bias - This algorithm updates the behavior of depth bias, by using rasterizer state. The result eliminates the scenarios where the calculated bias could be NaN.
- Resource limits - Resource indices are still required to be <= 32 bits, but resources can be larger than 4 GB.
- Rasterizer Precision
- MSAA Requirements
- Counters Reduced
- 1-Bit Format and Text Filter Removed

## Features Added in Previous Releases

For the list of the features added in previous releases, see the following topics:

- [What's New in the August 2009 Windows 7/Direct3D 11 SDK](#)
- [What's New in the November 2008 Direct3D 11 Technical Preview](#)

## Related topics

[What's new in Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11.1 Features

Article • 08/19/2020

The following functionality has been added in Direct3D 11.1, which is included with Windows 8, Windows RT, and Windows Server 2012. Partial support for [Direct3D 11.1](#) is available on Windows 7 and Windows Server 2008 R2 via the [Platform Update for Windows 7](#), which is available through the [Platform Update for Windows 7](#).

- Shader tracing and compiler enhancements
- Direct3D device sharing
- Check support of new Direct3D 11.1 features and formats
- Use HLSL minimum precision
- Specify user clip planes in HLSL on feature level 9 and higher
- Create larger constant buffers than a shader can access
- Use logical operations in a render target
- Force the sample count to create a rasterizer state
- Process video resources with shaders
- Extended support for shared Texture2D resources
- Change subresources with new copy options
- Discard resources and resource views
- Support a larger number of UAVs
- Bind a subrange of a constant buffer to a shader
- Retrieve the subrange of a constant buffer that is bound to a shader
- Clear all or part of a resource view
- Map SRVs of dynamic buffers with NO\_OVERWRITE
- Use UAVs at every pipeline stage
- Extended support for WARP devices
- Use Direct3D in Session 0 processes
- Support for shadow buffer on feature level 9
- Related topics

## Shader tracing and compiler enhancements

Direct3D 11.1 lets you use shader tracing to ensure that your code is performing as intended and if it isn't you can discover and remedy the problem. The Windows Software Development Kit (SDK) for Windows 8 contains HLSL compiler enhancements. Shader tracing and the HLSL compiler are implemented in D3dcompiler\_nn.dll.

The shader tracing API and the enhancements to the HLSL compiler consists of the following methods and functions.

- [ID3D11RefDefaultTrackingOptions::SetTrackingOptions](#)
- [ID3D11RefTrackingOptions::SetTrackingOptions](#)
- [ID3D11TracingDevice::SetShaderTrackingOptions](#)
- [ID3D11TracingDevice::SetShaderTrackingOptionsByType](#)
- [ID3D11ShaderTraceFactory::CreateShaderTrace](#)
- [ID3D11ShaderTrace::TraceReady](#)
- [ID3D11ShaderTrace::ResetTrace](#)
- [ID3D11ShaderTrace::GetTraceStats](#)
- [ID3D11ShaderTrace::PSSelectStamp](#)
- [ID3D11ShaderTrace::GetInitialRegisterContents](#)
- [ID3D11ShaderTrace::GetStep](#)
- [ID3D11ShaderTrace::GetWrittenRegister](#)
- [ID3D11ShaderTrace::GetReadRegister](#)
- [D3DCompile2](#)
- [D3DCompileFromFile](#)
- [D3DDisassemble11Trace](#)
- [D3DDisassembleRegion](#)
- [D3DGetTraceInstructionOffsets](#)
- [D3DReadFileToBlob](#)
- [D3DSetBlobPart](#)
- [D3DWriteBlobToFile](#)

The D3dcompiler.lib library requires D3dcompiler\_nn.dll. This DLL is not part of Windows 8; it is in the \bin folder of the Windows SDK for Windows 8 along with the Fxc.exe command-line version of the HLSL compiler.

#### Note

While you can use this library and DLL combination for development, you can't deploy Windows Store apps that use this combination. Therefore, you must instead compile HLSL shaders before you ship your Windows Store app. You can write HLSL compilation binaries to disk, or the compiler can generate headers with static byte arrays that contain the shader blob data. You use the **ID3DBlob** interface to access the blob data. To develop your Windows Store app, call **D3DCompile2** or **D3DCompileFromFile** to compile the raw HLSL source, and then feed the resulting blob data to Direct3D.

## Direct3D device sharing

Direct3D 11.1 enables Direct3D 10 APIs and Direct3D 11 APIs to use one underlying rendering device.

This Direct3D 11.1 feature consists of the following methods and interface.

- [ID3D11Device1::CreateDeviceContextState](#)
- [ID3D11DeviceContext1::SwapDeviceContextState](#)
- [ID3DDeviceContextState](#)

## Check support of new Direct3D 11.1 features and formats

Direct3D 11.1 lets you check for new features that the graphics driver might support and new ways that a format is supported on a device. Microsoft DirectX Graphics Infrastructure (DXGI) 1.2 also specifies new [DXGI\\_FORMAT](#) values.

This Direct3D 11.1 feature consists of the following API.

- [ID3D11Device::CheckFeatureSupport](#) with  
[D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#),  
[D3D11\\_FEATURE\\_DATA\\_ARCHITECTURE\\_INFO](#),  
[D3D11\\_FEATURE\\_DATA\\_D3D9\\_OPTIONS](#),  
[D3D11\\_FEATURE\\_DATA\\_SHADER\\_MIN\\_PRECISION\\_SUPPORT](#), and  
[D3D11\\_FEATURE\\_DATA\\_D3D9\\_SHADOW\\_SUPPORT](#) structures
- [ID3D11Device::CheckFormatSupport](#) with  
[D3D11\\_FORMAT\\_SUPPORT\\_DECODER\\_OUTPUT](#),  
[D3D11\\_FORMAT\\_SUPPORT\\_VIDEO\\_PROCESSOR\\_OUTPUT](#),  
[D3D11\\_FORMAT\\_SUPPORT\\_VIDEO\\_PROCESSOR\\_INPUT](#),  
[D3D11\\_FORMAT\\_SUPPORT\\_VIDEO\\_ENCODER](#), and  
[D3D11\\_FORMAT\\_SUPPORT2\\_OUTPUT\\_MERGER\\_LOGIC\\_OP](#)

## Use HLSL minimum precision

Starting with Windows 8, graphics drivers can implement minimum precision [HLSL scalar data types](#) by using any precision greater than or equal to their specified bit precision. When your HLSL minimum precision shader code is used on hardware that implements HLSL minimum precision, you use less memory bandwidth and as a result you also use less system power.

You can query for the minimum precision support that the graphics driver provides by calling [ID3D11Device::CheckFeatureSupport](#) with the

[D3D11\\_FEATURE\\_SHADER\\_MIN\\_PRECISION\\_SUPPORT](#) value. In this `ID3D11Device::CheckFeatureSupport` call, pass a pointer to a [D3D11\\_FEATURE\\_DATA\\_SHADER\\_MIN\\_PRECISION\\_SUPPORT](#) structure that `ID3D11Device::CheckFeatureSupport` fills with the minimum precision levels that the driver supports for the pixel shader stage and for other shader stages. 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.

You don't need to author multiple shaders that do and don't use minimum precision. Instead, create shaders with minimum precision, and the minimum precision variables behave at full 32-bit precision if the graphics driver reports that it doesn't support any minimum precision. For more info about HLSL minimum precision, see [Using HLSL minimum precision](#).

HLSL minimum precision shaders don't work on operating systems earlier than Windows 8.

## Specify user clip planes in HLSL on feature level 9 and higher

Starting with Windows 8, you can use the `clipplanes` function attribute in an HLSL [function declaration](#) rather than `SV_ClipDistance` to make your shader work on [feature level 9\\_x](#) as well as feature level 10 and higher. For more info, see [User clip planes on feature level 9 hardware](#).

## Create larger constant buffers than a shader can access

Direct3D 11.1 lets you create constant buffers that are larger than the maximum constant buffer size that a shader can access (4096 32-bit\*4-component constants – 64KB). Later, when you bind the buffers to the pipeline, for example, via `PSSetConstantBuffers` or `PSSetConstantBuffers1`, you can specify a range of buffers that the shader can access that fits within the 4096 limit.

Direct3D 11.1 updates the `ID3D11Device::CreateBuffer` method for this feature.

## Use logical operations in a render target

Direct3D 11.1 lets you use logical operations rather than blending in a render target. However, you can't mix logic operations with blending across multiple render targets.

This Direct3D 11.1 feature consists of the following API.

- [ID3D11Device1::CreateBlendState1](#) with [D3D11\\_LOGIC\\_OP](#)

## Force the sample count to create a rasterizer state

Direct3D 11.1 lets you specify a force sample count when you create a rasterizer state.

This Direct3D 11.1 feature consists of the following API.

- [ID3D11Device1::CreateRasterizerState1](#)

### Note

If you want to render with the sample count forced 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 ([D3D11\\_BIND\\_RENDER\\_TARGET](#)) and you forced the sample count to greater than 1, ensure that every render target has only a single sample.
- Don't operate the shader at sample frequency. Therefore, [ID3D11ShaderReflection::IsSampleFrequencyShader](#) returns FALSE.

Otherwise, rendering behavior is undefined. For info about how to configure depth-stencil, see [Configuring Depth-Stencil Functionality](#).

## Process video resources with shaders

Direct3D 11.1 lets you create views (SRV/RTV/UAV) to video resources so that Direct3D shaders can process those video resources. The format of an underlying video resource restricts the formats that the view can use. The [DXGI\\_FORMAT](#) enumeration contains

new video resource format values. These [DXGI\\_FORMAT](#) values specify the valid view formats that you can create and how the Direct3D 11.1 runtime maps the view. You can create multiple views of different parts of the same surface, and depending on the format, the sizes of the views can differ from each other.

Direct3D 11.1 updates the following methods for this feature.

- [ID3D11Device::CreateShaderResourceView](#)
- [ID3D11Device::CreateRenderTargetView](#)
- [ID3D11Device::CreateUnorderedAccessView](#)

## Extended support for shared Texture2D resources

Direct3D 11.1 guarantees that you can share Texture2D resources that you created with particular resource types and formats. To share Texture2D resources, use the [D3D11\\_RESOURCE\\_MISC\\_SHARED](#), [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_KEYEDMUTEX](#), or a combination of the [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_KEYEDMUTEX](#) and [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_NTHANDLE](#) (new for Windows 8) flags when you create those resources.

Direct3D 11.1 guarantees that you can share Texture2D resources that you created with these [DXGI\\_FORMAT](#) values:

- [DXGI\\_FORMAT\\_R8G8B8A8\\_UNORM](#)
- [DXGI\\_FORMAT\\_R8G8B8A8\\_UNORM\\_SRGB](#)
- [DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM](#)
- [DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM\\_SRGB](#)
- [DXGI\\_FORMAT\\_B8G8R8X8\\_UNORM](#)
- [DXGI\\_FORMAT\\_B8G8R8X8\\_UNORM\\_SRGB](#)
- [DXGI\\_FORMAT\\_R10G10B10A2\\_UNORM](#)
- [DXGI\\_FORMAT\\_R16G16B16A16\\_FLOAT](#)

In addition, Direct3D 11.1 guarantees that you can share Texture2D resources that you created with a mipmap level of 1, array size of 1, bind flags of [D3D11\\_BIND\\_SHADER\\_RESOURCE](#) and [D3D11\\_BIND\\_RENDER\\_TARGET](#) combined, usage default ([D3D11\\_USAGE\\_DEFAULT](#)), and only these [D3D11\\_RESOURCE\\_MISC\\_FLAG](#) values:

- [D3D11\\_RESOURCE\\_MISC\\_SHARED](#)
- [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_KEYEDMUTEX](#)
- [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_NTHANDLE](#)
- [D3D11\\_RESOURCE\\_MISC\\_GDI\\_COMPATIBLE](#)

Direct3D 11.1 lets you share a greater variety of Texture2D resource types and formats. You can query for whether the graphics driver and hardware support extended Texture2D resource sharing by calling [ID3D11Device::CheckFeatureSupport](#) with the [D3D11\\_FEATURE\\_D3D11\\_OPTIONS](#) value. In this [ID3D11Device::CheckFeatureSupport](#) call, pass a pointer to a [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) structure. [ID3D11Device::CheckFeatureSupport](#) sets the [ExtendedResourceSharing](#) member of [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) to TRUE if the hardware and driver support extended Texture2D resource sharing.

If [ID3D11Device::CheckFeatureSupport](#) returns TRUE in [ExtendedResourceSharing](#), you can share resources that you created with these [DXGI\\_FORMAT](#) values:

- [DXGI\\_FORMAT\\_R32G32B32A32\\_TYPELESS](#)
- [DXGI\\_FORMAT\\_R32G32B32A32\\_FLOAT](#)
- [DXGI\\_FORMAT\\_R32G32B32A32\\_UINT](#)
- [DXGI\\_FORMAT\\_R32G32B32A32\\_SINT](#)
- [DXGI\\_FORMAT\\_R16G16B16A16\\_TYPELESS](#)
- [DXGI\\_FORMAT\\_R16G16B16A16\\_FLOAT](#)
- [DXGI\\_FORMAT\\_R16G16B16A16\\_UNORM](#)
- [DXGI\\_FORMAT\\_R16G16B16A16\\_UINT](#)
- [DXGI\\_FORMAT\\_R16G16B16A16\\_SNORM](#)
- [DXGI\\_FORMAT\\_R16G16B16A16\\_SINT](#)
- [DXGI\\_FORMAT\\_R10G10B10A2\\_UNORM](#)
- [DXGI\\_FORMAT\\_R10G10B10A2\\_UINT](#)
- [DXGI\\_FORMAT\\_R8G8B8A8\\_TYPELESS](#)
- [DXGI\\_FORMAT\\_R8G8B8A8\\_UNORM](#)
- [DXGI\\_FORMAT\\_R8G8B8A8\\_UNORM\\_SRGB](#)
- [DXGI\\_FORMAT\\_R8G8B8A8\\_UINT](#)
- [DXGI\\_FORMAT\\_R8G8B8A8\\_SNORM](#)
- [DXGI\\_FORMAT\\_R8G8B8A8\\_SINT](#)
- [DXGI\\_FORMAT\\_B8G8R8A8\\_TYPELESS](#)
- [DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM](#)
- [DXGI\\_FORMAT\\_B8G8R8X8\\_UNORM](#)
- [DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM\\_SRGB](#)
- [DXGI\\_FORMAT\\_B8G8R8X8\\_TYPELESS](#)
- [DXGI\\_FORMAT\\_B8G8R8X8\\_UNORM\\_SRGB](#)
- [DXGI\\_FORMAT\\_R32\\_TYPELESS](#)
- [DXGI\\_FORMAT\\_R32\\_FLOAT](#)
- [DXGI\\_FORMAT\\_R32\\_UINT](#)
- [DXGI\\_FORMAT\\_R32\\_SINT](#)
- [DXGI\\_FORMAT\\_R16\\_TYPELESS](#)

- DXGI\_FORMAT\_R16\_FLOAT
- DXGI\_FORMAT\_R16\_UNORM
- DXGI\_FORMAT\_R16\_UINT
- DXGI\_FORMAT\_R16\_SNORM
- DXGI\_FORMAT\_R16\_SINT
- DXGI\_FORMAT\_R8\_TYPELESS
- DXGI\_FORMAT\_R8\_UNORM
- DXGI\_FORMAT\_R8\_UINT
- DXGI\_FORMAT\_R8\_SNORM
- DXGI\_FORMAT\_R8\_SINT
- DXGI\_FORMAT\_A8\_UNORM
- DXGI\_FORMAT\_AYUV
- DXGI\_FORMAT\_YUY2
- DXGI\_FORMAT\_NV12
- DXGI\_FORMAT\_NV11
- DXGI\_FORMAT\_P016
- DXGI\_FORMAT\_P010
- DXGI\_FORMAT\_Y216
- DXGI\_FORMAT\_Y210
- DXGI\_FORMAT\_Y416
- DXGI\_FORMAT\_Y410

If [ID3D11Device::CheckFeatureSupport](#) returns TRUE in `ExtendedResourceSharing`, you can share resources that you created with these features and flags:

- [D3D11\\_USAGE\\_DEFAULT](#)
- [D3D11\\_BIND\\_SHADER\\_RESOURCE](#)
- [D3D11\\_BIND\\_RENDER\\_TARGET](#)
- [D3D11\\_RESOURCE\\_MISC\\_GENERATE\\_MIPS](#)
- [D3D11\\_BIND\\_UNORDERED\\_ACCESS](#)
- Mipmap levels (one or more levels) in the 2D texture resources (specified in the `MipLevels` member of [D3D11\\_TEXTURE2D\\_DESC](#))
- Arrays of 2D texture resources (one or more textures) (specified in the `ArraySize` member of [D3D11\\_TEXTURE2D\\_DESC](#))
- [D3D11\\_BIND\\_DECODER](#)
- [D3D11\\_RESOURCE\\_MISC\\_RESTRICTED\\_CONTENT](#)
- [D3D11\\_RESOURCE\\_MISC\\_RESTRICT\\_SHARED\\_RESOURCE](#)
- [D3D11\\_RESOURCE\\_MISC\\_RESTRICT\\_SHARED\\_RESOURCE\\_DRIVER](#)

 **Note**

When `ExtendedResourceSharing` is TRUE, you have more flexibility when you specify bind flags for sharing Texture2D resources. Not only does the graphics driver and hardware support more bind flags but also more possible combinations of bind flags. For example, you can specify just `D3D11_BIND_RENDER_TARGET` or no bind flags, and so on.

Even if `ID3D11Device::CheckFeatureSupport` returns TRUE in `ExtendedResourceSharing`, you still can't share resources that you created with these features and flags:

- `D3D11_BIND_DEPTH_STENCIL`
- 2D texture resources in multisample antialiasing (MSAA) mode (specified with `DXGI_SAMPLE_DESC`)
- `D3D11_RESOURCE_MISC_RESOURCE_CLAMP`
- `D3D11_USAGE_IMMUTABLE`, `D3D11_USAGE_DYNAMIC`, or `D3D11_USAGE_STAGING` (that is, mappable)
- `D3D11_RESOURCE_MISC_TEXTURECUBE`

## Change subresources with new copy options

Direct3D 11.1 lets you use new copy flags to copy and update subresources. When you copy a subresource, the source and destination resources can be identical and the source and destination regions can overlap.

This Direct3D 11.1 feature consists of the following API.

- `ID3D11DeviceContext1::CopySubresourceRegion1`
- `ID3D11DeviceContext1::UpdateSubresource1`

## Discard resources and resource views

Direct3D 11.1 lets you discard resources and views of resources from the device context. This new functionality informs the GPU that existing content in resources or resource views are no longer needed.

This Direct3D 11.1 feature consists of the following API.

- `ID3D11DeviceContext1::DiscardResource`
- `ID3D11DeviceContext1::DiscardView`
- `ID3D11DeviceContext1::DiscardView1`

# Support a larger number of UAVs

Direct3D 11.1 lets you use a larger number of UAVs when you bind resources to the [output-merger stage](#) and when you set an array of views for an unordered resource.

Direct3D 11.1 updates the following methods for this feature.

- [ID3D11DeviceContext::OMSetRenderTargetsAndUnorderedAccessViews](#)
- [ID3D11DeviceContext::CSSetUnorderedAccessViews](#)

## Bind a subrange of a constant buffer to a shader

Direct3D 11.1 lets you bind a subrange of a constant buffer for a shader to access. You can supply a larger constant buffer and specify the subrange that the shader can use.

This Direct3D 11.1 feature consists of the following API.

- [ID3D11DeviceContext1::CSSetConstantBuffers1](#)
- [ID3D11DeviceContext1::DSSetConstantBuffers1](#)
- [ID3D11DeviceContext1::GSSetConstantBuffers1](#)
- [ID3D11DeviceContext1::HSSetConstantBuffers1](#)
- [ID3D11DeviceContext1::PSSetConstantBuffers1](#)
- [ID3D11DeviceContext1::VSSetConstantBuffers1](#)

## Retrieve the subrange of a constant buffer that is bound to a shader

Direct3D 11.1 lets you retrieve the subrange of a constant buffer that is bound to a shader.

This Direct3D 11.1 feature consists of the following API.

- [ID3D11DeviceContext1::CSGetConstantBuffers1](#)
- [ID3D11DeviceContext1::DSGetConstantBuffers1](#)
- [ID3D11DeviceContext1::GSGetConstantBuffers1](#)
- [ID3D11DeviceContext1::HSGetConstantBuffers1](#)
- [ID3D11DeviceContext1::PSGetConstantBuffers1](#)
- [ID3D11DeviceContext1::VSGetConstantBuffers1](#)

## Clear all or part of a resource view

Direct3D 11.1 lets you clear a resource view (RTV, UAV, or any video view of a Texture2D surface). You apply the same color value to all parts of the view.

This Direct3D 11.1 feature consists of the following API.

- [ID3D11DeviceContext::ClearView](#)

## Map SRVs of dynamic buffers with NO\_OVERWRITE

Direct3D 11.1 lets you map shader resource views (SRV) of dynamic buffers with D3D11\_MAP\_WRITE\_NO\_OVERWRITE. The Direct3D 11 and earlier runtimes limited mapping to vertex or index buffers.

Direct3D 11.1 updates the [ID3D11DeviceContext::Map](#) method for this feature.

## Use UAVs at every pipeline stage

Direct3D 11.1 lets you use the following shader model 5.0 instructions at all shader stages that were previously used in just pixel shaders and compute shaders.

- [dcl\\_uav\\_typed](#)
- [dcl\\_uav\\_raw](#)
- [dcl\\_uav\\_structured](#)
- [ld\\_raw](#)
- [ld\\_structured](#)
- [ld\\_uav\\_typed](#)
- [store\\_raw](#)
- [store\\_structured](#)
- [store\\_uav\\_typed](#)
- [sync\\_uglobal](#)
- All atomics and immediate atomics (for example, [atomic\\_and](#) and [imm\\_atomic\\_and](#))

Direct3D 11.1 updates the following methods for this feature.

- [ID3D11DeviceContext::CreateDomainShader](#)
- [ID3D11DeviceContext::CreateGeometryShader](#)
- [ID3D11DeviceContext::CreateGeometryShaderWithStreamOutput](#)
- [ID3D11DeviceContext::CreateHullShader](#)

- [ID3D11DeviceContext::CreateVertexShader](#)

These instructions existed in Direct3D 11.0 in ps\_5\_0 and cs\_5\_0. Because Direct3D 11.1 makes UAVs available at all shader stages, these instructions are available at all shader stages.

If you pass compiled shaders (VS/HS/DS/HS) that use any of these instructions to a create-shader function, like [CreateVertexShader](#), on devices that don't support UAVs at every stage (including existing drivers that are not implemented with this feature), the create-shader function fails. The create-shader function also fails if the shader tries to use a UAV slot beyond the set of UAV slots that the hardware supports.

The UAVs that are referenced by these instructions are shared across all pipeline stages. For example, a UAV that is bound at slot 0 at the [output-merger stage](#) is available at slot 0 to VS/HS/DS/GS/PS.

UAV accesses that you issue from within or across shader stages that execute within a given Draw\*() or that you issue from the compute shader within a Dispatch\*() aren't guaranteed to finish in the order in which you issued them. But all UAV accesses finish at the end of the Draw\*() or Dispatch\*().

## Extended support for WARP devices

Direct3D 11.1 extends support for [WARP](#) devices, which you create by passing [D3D\\_DRIVER\\_TYPE\\_WARP](#) in the *DriverType* parameter of [D3D11CreateDevice](#).

Starting with Direct3D 11.1 WARP devices support:

- All Direct3D [feature levels](#) from 9.1 through to 11.1
- [Compute shaders](#) and [tessellation](#)
- Shared surfaces. That is, you can fully share surfaces between WARP devices, as well as between WARP devices in different processes.

WARP devices don't support these optional features:

- [doubles](#)
- [video encode or decode](#)
- [minimum precision shader support](#)

When you run a virtual machine (VM), Hyper-V, with your graphics processing unit (GPU) disabled, or without a display driver, you get a WARP device whose friendly name is "Microsoft Basic Display Adapter." This WARP device can run the full Windows experience, which includes DWM, Internet Explorer, and Windows Store apps. This WARP

device also supports running legacy apps that use [DirectDraw](#) or running apps that use Direct3D 3 through Direct3D 11.1.

## Use Direct3D in Session 0 processes

Starting with Windows 8 and Windows Server 2012, you can use most of the Direct3D APIs in Session 0 processes.

### Note

These output, window, swap chain, and presentation-related APIs are not available in Session 0 processes because they don't apply to the Session 0 environment:

- `IDXGIFactory::CreateSwapChain`
- `IDXGIFactory::GetWindowAssociation`
- `IDXGIFactory::MakeWindowAssociation`
- `IDXGIAdapter::EnumOutputs`
- `ID3D11Debug::SetFeatureMask`
- `ID3D11Debug::SetPresentPerRenderOpDelay`
- `ID3D11Debug::SetSwapChain`
- `ID3D10Debug::SetFeatureMask`
- `ID3D10Debug::SetPresentPerRenderOpDelay`
- `ID3D10Debug::SetSwapChain`
- `D3D10CreateDeviceAndSwapChain`
- `D3D10CreateDeviceAndSwapChain1`
- `D3D11CreateDeviceAndSwapChain`

If you call one of the preceding APIs in a Session 0 process, it returns `DXGI_ERROR_NOT_CURRENTLY_AVAILABLE`.

## Support for shadow buffer on feature level 9

Use a subset of Direct3D 10\_0+ shadow buffer features to implement shadow effects on [feature level 9\\_x](#). For info about what you need to do to implement shadow effects on feature level 9\_x, see [Implementing shadow buffers for Direct3D feature level 9](#).

# Related topics

[What's new in Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11.2 Features

Article • 08/19/2020

The following functionality has been added in Direct3D 11.2, which is included with Windows 8.1, Windows RT 8.1, and Windows Server 2012 R2.

- [Tiled resources](#)
  - [Check tiled resources support](#)
- [Extended support for WARP devices](#)
- [Annotate graphics commands](#)
- [HLSL shader linking](#)
  - [Function linking graph \(FLG\)](#)
- [Inbox HLSL compiler](#)
- [Related topics](#)

## Tiled resources

Direct3D 11.2 lets you create tiled resources that can be thought of as large logical resources that use small amounts of physical memory. Tiled resources are useful (for example) with terrain in games, and app UI.

Tiled resources are created by specifying the [D3D11\\_RESOURCE\\_MISC\\_TILED](#) flag. To work with tiled resource, use these API:

- [ID3D11Device2::GetResourceTiling](#)
- [ID3D11DeviceContext2::UpdateTiles](#)
- [ID3D11DeviceContext2::UpdateTileMappings](#)
- [ID3D11DeviceContext2::CopyTiles](#)
- [ID3D11DeviceContext2::CopyTileMappings](#)
- [ID3D11DeviceContext2::ResizeTilePool](#)
- [ID3D11DeviceContext2::TiledResourceBarrier](#)
- [D3D11\\_DEBUG\\_FEATURE\\_DISABLE\\_TILED\\_RESOURCE\\_MAPPING\\_TRACKING\\_AND\\_VALIDATION](#) flag with [ID3D11Debug::SetFeatureMask](#)

For more info about tiled resources, see [Tiled resources](#).

## Check tiled resources support

Before you use tiled resources, you need to find out if the device supports tiled resources. Here's how you check for support for tiled resources:

C++

```

HRESULT hr = D3D11CreateDevice(
    nullptr,                                // Specify nullptr to use the default adapter.
    D3D_DRIVER_TYPE_HARDWARE,                // Create a device using the hardware graphics
    driver,                                 // Should be 0 unless the driver is
    0,                                      // Set debug and Direct2D compatibility flags.
    D3D_DRIVER_TYPE_SOFTWARE,               // List of feature levels this app can support.
    creationFlags,                         // Size of the list above.
    &featureLevels,                        // Always set this to D3D11_SDK_VERSION for
    ARRSIZE(featureLevels),                // Windows Store apps.
    D3D11_SDK_VERSION,                     // Returns the Direct3D device created.
    &device,                               // Returns feature level of device created.
    &context                               // Returns the device immediate context.
);

if (SUCCEEDED(hr))
{
    D3D11_FEATURE_DATA_D3D11_OPTIONS1 featureData;
    DX::ThrowIfFailed(
        device->CheckFeatureSupport(D3D11_FEATURE_D3D11_OPTIONS1, &featureData,
        sizeof(featureData))
    );

    m_tiledResourcesTier = featureData.TiledResourcesTier;
}

```

## Extended support for WARP devices

Direct3D 11.2 extends support for [WARP](#) devices, which you create by passing [D3D\\_DRIVER\\_TYPE\\_WARP](#) in the *DriverType* parameter of [D3D11CreateDevice](#). The WARP software renderer in Direct3D 11.2 adds full support for Direct3D [feature level](#) 11\_1, including [tiled resources](#), [IDXGIDevice3::Trim](#), shared BCn surfaces, minblend, and map default. [Double](#) support in HLSL shaders has also been enabled along with support for 16x MSAA.

## Annotate graphics commands

Direct3D 11.2 lets you annotate graphics commands with these API:

- [ID3D11DeviceContext2::IsAnnotationEnabled](#)
- [ID3D11DeviceContext2::BeginEventInt](#)
- [ID3D11DeviceContext2::SetMarkerInt](#)
- [ID3D11DeviceContext2::EndEvent](#)

## HLSL shader linking

Windows 8.1 adds separate compilation and linking of HLSL shaders, which allows graphics programmers to create precompiled HLSL functions, package them into libraries, and link

them into full shaders at run-time. This is essentially an equivalent of C/C++ separate compilation, libraries, and linking, and it allows programmers to compose precompiled HLSL code when more information becomes available to finalize the computation. For more info about how to use shader linking, see [Using shader linking](#).

Complete these steps to create a final shader using dynamic linkage at run time.

### To create and use shader linking

1. Create a [ID3D11Linker](#) linker object, which represents a linking context. A single context can't be used to produce multiple shaders; a linking context is used to produce a single shader and then the linking context is thrown away.
2. Use [D3DLoadModule](#) to load and set libraries from their library blobs.
3. Use [D3DLoadModule](#) to load and set an entry shader blob, or create an [FLG shader](#).
4. Use [ID3D11Module::CreateInstance](#) to create [ID3D11ModuleInstance](#) objects, then call functions on these objects to rebind resources to their final slots.
5. Add the libraries to the linker, then call [ID3D11Linker::Link](#) to produce final shader byte code that can then be loaded and used in the runtime just like a fully precompiled and linked shader.

## Function linking graph (FLG)

Windows 8.1 also adds the Function Linking Graph (FLG). You can use FLG to construct shaders that consist of a sequence of precompiled function invocations that pass values to each other. When using the FLG, there is no need to write HLSL and invoke the HLSL compiler. Instead, the shader structure is specified programmatically using C++ API calls. FLG nodes represent input and output signatures and invocations of precompiled library functions. The order of registering the function-call nodes defines the sequence of invocations. The input signature node must be specified first, while the output signature node must be specified last. FLG edges define how values are passed from one node to another. The data types of passed values must be the same; there is no implicit type conversion. Shape and swizzling rules follow the HLSL behavior and values can only be passed forward in this sequence. For info on the FLG API, see [ID3D11FunctionLinkingGraph](#).

## Inbox HLSL compiler

The HLSL compiler is now inbox on Windows 8.1 and later. Now, most APIs for shader programming can be used in Windows Store apps that are built for Windows 8.1 and later. Many APIs for shader programming couldn't be used in Windows Store apps that were built for Windows 8; the reference pages for these APIs were marked with a note. But some shader APIs (for example, [D3DCompileFromFile](#)) can still only be used to develop Windows Store apps, and not in apps that you submit to the Windows Store; the reference pages for these APIs are still marked with a note.

## Related topics

[What's new in Direct3D 11](#)

---

## Feedback

Was this page helpful?



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

# Direct3D 11.3 Features

Article • 11/04/2020

The following sections describe the functionality has been added in Direct3D 11.3. These features are also available in Direct3D 12.

## In this section

Topic	Description
Conservative Rasterization	Conservative rasterization adds some certainty to pixel rendering, which is helpful in particular to collision detection algorithms.
Default Texture Mapping	The use of default texture mapping reduces copying and memory usage while sharing image data between the GPU and the CPU. However, it should only be used in specific situations. The standard swizzle layout avoids copying or swizzling data in multiple layouts.
Rasterizer Order Views	Rasterizer ordered views (ROVs) allow pixel shader code to mark UAV bindings with a declaration that alters the normal requirements for the order of graphics pipeline results for UAVs. This enables Order Independent Transparency (OIT) algorithms to work, which give much better rendering results when multiple transparent objects are in line with each other in a view.
Shader Specified Stencil Reference Value	Enabling pixel shaders to output the Stencil Reference Value, rather than using the API-specified one, enables a very fine granular control over stencil operations.
Typed Unordered Access View Loads	Unordered Access View (UAV) Typed Load is the ability for a shader to read from a UAV with a specific DXGI_FORMAT.
Unified Memory Architecture	Querying for whether Unified Memory Architecture (UMA) is supported can help determine how to handle some resources.
Volume Tiled Resources	Volume (3D) textures can be used as tiled resources, noting that tile resolution is three-dimensional.

## Related topics

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Direct3D 11.3 Conservative Rasterization

Article • 08/19/2020

Conservative rasterization adds some certainty to pixel rendering, which is helpful in particular to collision detection algorithms.

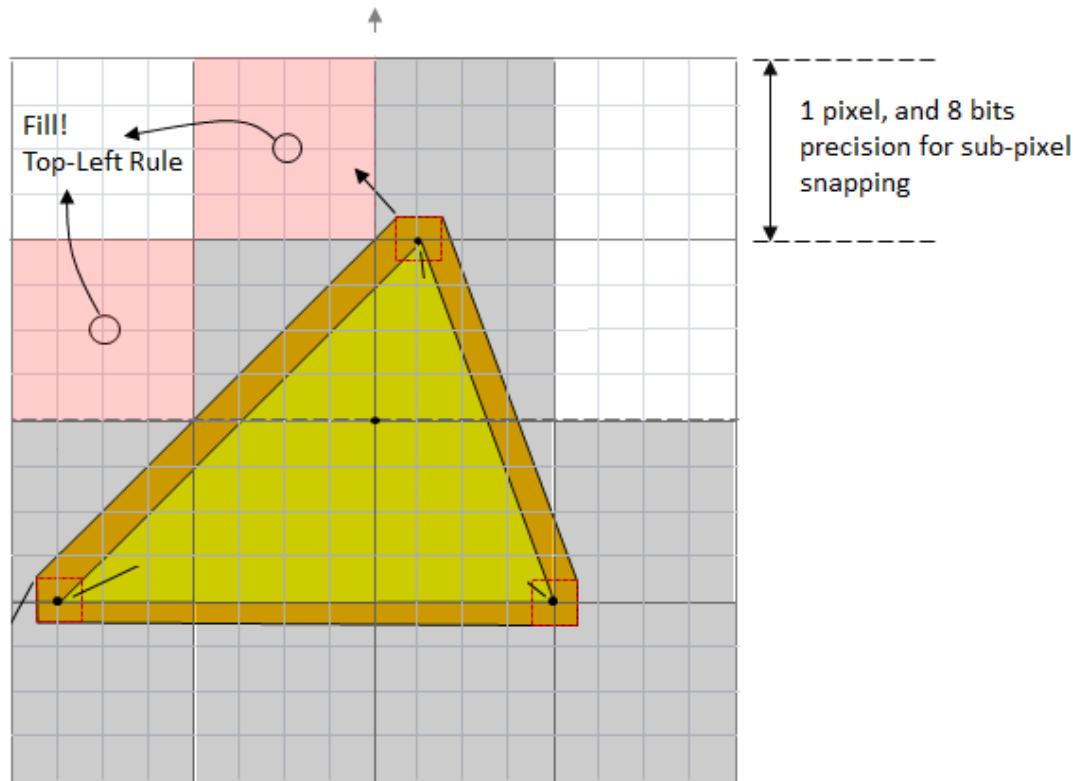
- [Overview](#)
- [Interactions with the pipeline](#)
- [Implementation details](#)
- [API summary](#)
- [Related topics](#)

## Overview

Conservative rasterization means that all pixels that are at least partially covered by a rendered primitive are rasterized, which means that the pixel shader is invoked. Normal behavior is sampling, which is not used if conservative rasterization is enabled.

Conservative rasterization is useful in a number of situations, including for certainty in collision detection, occlusion culling, and visibility detection.

For example, the following figure shows a green triangle rendered using conservative rasterization. The brown area is known as an "uncertainty region" - a region where rounding errors and other issues add some uncertainty to the exact dimensions of the triangle. The red triangles at each vertex show how the uncertainty region is calculated. The large gray squares show the pixels that will be rendered. The pink squares show pixels rendered using the "top-left rule", which comes into play as the edge of the triangle crosses the edge of the pixels. There can be false positives (pixels set that should not have been) which the system will normally but not always cull.



## Interactions with the pipeline

For many details on how conservative rasterization interacts with the graphics pipeline, refer to [D3D12 Conservative rasterization](#).

## Implementation details

The type of rasterization supported in Direct3D 12 is sometimes referred to as "overestimated conservative rasterization". There is also the concept of "underestimated conservative rasterization", which means that only pixels that are fully covered by a rendered primitive are rasterized. Underestimated conservative rasterization information is available through the pixel shader through the use of input coverage data, and only overestimated conservative rasterization is available as a rasterizing mode.

If any part of a primitive overlaps a pixel, then that pixel is considered covered and is then rasterized. When an edge or corner of a primitive falls along the edge or corner of a pixel, the application of the "top-left rule" is implementation-specific. However, for implementations that support degenerate triangles, a degenerate triangle along an edge or corner must cover at least one pixel.

Conservative rasterization implementations can vary on different hardware, and do produce false positives, meaning that they can incorrectly decide that pixels are covered. This can occur because of implementation-specific details like primitive growing or

snapping errors inherent in the fixed-point vertex coordinates used in rasterization. The reason false positives (with respect to fixed point vertex coordinates) are valid is because some amount of false positives are needed to allow an implementation to do coverage evaluation against post-snapped vertices (i.e. vertex coordinates that have been converted from floating point to the 16.8 fixed-point used in the rasterizer), but honor the coverage produced by the original floating point vertex coordinates.

Conservative rasterization implementations do not produce false negatives with respect to the floating-point vertex coordinates for non-degenerate post-snap primitives: if any part of a primitive overlaps any part of a pixel, then that pixel is rasterized.

Triangles that are degenerate (duplicate indices in an index buffer or collinear in 3D), or become degenerate after fixed-point conversion (collinear vertices in the rasterizer), may or may not be culled; both are valid behaviors. Degenerate triangles must be considered back facing, so if a specific behavior is required by an application, it can use back-face culling or test for front facing. Degenerate triangles use the values assigned to Vertex 0 for all interpolated values.

There are three tiers of hardware support, in addition to the possibility that the hardware does not support this feature.

- Tier 1 supports 1/2 pixel uncertainty regions, and no post-snap degenerates. This is good for tiled rendering, a texture atlas, light map generation and sub-pixel shadow maps.
- Tier 2 adds post-snap degenerates, and 1/256 uncertainty regions. It also adds support for CPU-based algorithm acceleration (such as voxelization).
- Tier 3 adds 1/512 uncertainty regions, inner input coverage and supports occlusion culling. The input coverage adds the new value `SV_InnerCoverage` to High Level Shading Language (HLSL). This is a 32-bit scalar integer that can be specified on input to a pixel shader, and represents the underestimated conservative rasterization information (that is, whether a pixel is guaranteed-to-be-fully covered).

## API summary

The following methods, structures, enums, and helper classes reference conservative rasterization:

- [D3D11\\_RASTERIZER\\_DESC2](#) : structure holding the rasterizer description, noting the CD3D12\_RASTERIZER\_DESC2 helper class for creating rasterizer descriptions.
- [D3D11\\_CONSERVATIVE\\_RASTERIZATION\\_MODE](#) : enum values for the mode (on or off).

- [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#) : structure holding the tier of support.
- [D3D11\\_CONSERVATIVE\\_RASTERIZATION\\_TIER](#) : enum values for each tier of support by the hardware.
- [ID3D11Device::CheckFeatureSupport](#) : method to access the supported features.

## Related topics

- [Direct3D 11.3 Features](#)
- 

## Feedback

Was this page helpful?



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

# Default Texture Mapping

Article • 08/23/2019

The use of default texture mapping reduces copying and memory usage while sharing image data between the GPU and the CPU. However, it should only be used in specific situations. The standard swizzle layout avoids copying or swizzling data in multiple layouts.

- [Overview](#)
- [D3D11.3 APIs](#)
- [Related topics](#)

## Overview

Mapping default textures should not be the first choice for developers. Developers should code in a discrete-GPU friendly way first (that is, do not have any CPU access for the majority of textures and upload with [CopySubresourceRegion1](#)). But, for certain cases, the CPU and GPU may interact so frequently on the same data, that mapping default textures becomes helpful to save power, or to speed up a particular design on particular adapters or architectures. Applications should detect these cases and optimize out the unnecessary copies.

In D3D11.3, textures created with D3D11\_TEXTURE\_LAYOUT\_UNDEFINED (one member of the [D3D11\\_TEXTURE\\_LAYOUT](#) enum) and no CPU access are the most efficient for frequent GPU rendering and sampling. When performance testing, those textures should be compared against D3D11\_TEXTURE\_LAYOUT\_UNDEFINED with CPU access, and D3D11\_TEXTURE\_LAYOUT\_64K\_STANDARD\_SWIZZLE with CPU access, and D3D11\_TEXTURE\_LAYOUT\_ROW\_MAJOR for cross-adapter support.

Using D3D11\_TEXTURE\_LAYOUT\_UNDEFINED with CPU access enables the methods [WriteToSubresource](#), [ReadFromSubresource](#), [Map](#) (precluding application access to pointer), and [Unmap](#); but can sacrifice efficiency of GPU access. Using D3D11\_TEXTURE\_LAYOUT\_64K\_STANDARD\_SWIZZLE with CPU access enables [WriteToSubresource](#), [ReadFromSubresource](#), [Map](#) (which returns a valid pointer to application), and [Unmap](#). It can also sacrifice efficiency of GPU access more than D3D11\_TEXTURE\_LAYOUT\_UNDEFINED with CPU access.

In general, applications should create the majority of textures as GPU-only-accessible, and with D3D11\_TEXTURE\_LAYOUT\_UNDEFINED.

Previous to the mapping default textures feature, there was only one standardized layout for multi-dimensional data: "linear", also known as "row-major". Applications should avoid USAGE\_STAGING and USAGE\_DYNAMIC textures when map default is available. The USAGE\_STAGING and USAGE\_DYNAMIC textures use the linear layout.

D3D11.3 (and D3D12) introduce a standard multi-dimensional data layout. This is done to enable multiple processing units to operate on the same data without copying the data or swizzling the data between multiple layouts. A standardized layout enables efficiency gains through network effects and allows algorithms to make short-cuts assuming a particular pattern.

Note though that this standard swizzle is a hardware feature, and may not be supported by all GPUs.

## D3D11.3 APIs

Unlike D3D12, D3D11.3 does not supports texture mapping by default, so you need to query [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#). Standard swizzle will also need to be queried for with a call to [ID3D11Device::CheckFeatureSupport](#) and checking the `StandardSwizzle64KBSupported` field of [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#).

The following APIs reference texture mapping:

### Enums

- [D3D11\\_TEXTURE\\_LAYOUT](#) : controls the swizzle pattern of default textures and enable map support on default textures.
- [D3D11\\_FEATURE](#) : references [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#).
- [D3D11\\_TILE\\_COPY\\_FLAG](#) : contains flags to copy swizzled tiled resources to and from linear buffers.

### Structures

- [D3D11\\_TEXTURE2D\\_DESC1](#) : describes a 2D texture. Note the `CD3D11_TEXTURE2D_DESC1` helper structure.
- [D3D11\\_TEXTURE3D\\_DESC1](#) : describes a 3D texture. Note the `CD3D11_TEXTURE3D_DESC1` helper structure.

### Methods

- [ID3D11Device3::CreateTexture2D1](#) : creates an array of 2D textures.
- [ID3D11Device3::CreateTexture3D1](#) : creates a single 3D texture.

- [ID3D11Device3::WriteToSubresource](#) : copies data into a D3D11\_USAGE\_DEFAULT texture which was mapped using [Map](#).
- [ID3D11Device3::ReadFromSubresource](#) : copies data from a D3D11\_USAGE\_DEFAULT texture which was mapped using [Map](#).
- [ID3D11DeviceContext::Map](#) : gets a pointer to the data contained in a subresource, and denies the GPU access to that subresource.
- [ID3D11DeviceContext::Unmap](#) : invalidates the pointer to a resource and reenables the GPU's access to that resource.
- [ID3D11Texture2D1::GetDesc1](#) : gets the properties of a 2D texture resource.
- [ID3D11Texture3D1::GetDesc1](#) : gets the properties of a 3D texture resource.

## Related topics

[Direct3D 11.3 Features](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Rasterizer Order Views

Article • 08/19/2020

Rasterizer ordered views (ROVs) allow pixel shader code to mark UAV bindings with a declaration that alters the normal requirements for the order of graphics pipeline results for UAVs. This enables Order Independent Transparency (OIT) algorithms to work, which give much better rendering results when multiple transparent objects are in line with each other in a view.

- [Overview](#)
- [Implementation details](#)
- [API summary](#)
- [Related topics](#)

## Overview

Standard graphics pipelines may have trouble correctly compositing together multiple textures that contain transparency. Objects such as wire fences, smoke, fire, vegetation, and colored glass use transparency to get the desired effect. Problems arise when multiple textures that contain transparency are in line with each other (smoke in front of a fence in front of a glass building containing vegetation, as an example). Rasterizer ordered views (ROVs) enable the underlying OIT algorithms to use features of the hardware to try to resolve the transparency order correctly. Transparency is handled by the pixel shader.

Rasterizer ordered views (ROVs) allow pixel shader code to mark UAV bindings with a declaration that alters the normal requirements for the order of graphics pipeline results for UAVs.

ROVs guarantee the order of UAV accesses for any pair of overlapping pixel shader invocations. In this case “overlapping” means that the invocations are generated by the same draw calls and share the same pixel coordinate when in pixel-frequency execution mode, and the same pixel and sample coordinate in sample-frequency mode.

The order in which overlapping ROV accesses of pixel shader invocations are executed is identical to the order in which the geometry is submitted. This means that, for overlapping pixel shader invocations, ROV writes performed by a pixel shader invocation must be available to be read by a subsequent invocation and must not affect reads by a previous invocation. ROV reads performed by a pixel shader invocation must reflect writes by a previous invocation and must not reflect writes by a subsequent invocation.

This is important for UAVs because they are explicitly omitted from the output-invariance guarantees normally set by the fixed order of graphics pipeline results.

## Implementation details

Rasterizer ordered views (ROVs) are declared with the following new High Level Shader Language (HLSL) objects, and are only available to the pixel shader:

- `RasterizerOrderedBuffer`
- `RasterizerOrderedByteAddressBuffer`
- `RasterizerOrderedStructuredBuffer`
- `RasterizerOrderedTexture1D`
- `RasterizerOrderedTexture1DArray`
- `RasterizerOrderedTexture2D`
- `RasterizerOrderedTexture2DArray`
- `RasterizerOrderedTexture3D`

Use these objects in the same manner as other UAV objects (such as `RWBuffer` etc.).

## API summary

ROVs are an HLSL-only construct that applies different behavior semantics to UAVs. All APIs relevant to UAVs are also relevant to ROVs. Note that the following method, structures, and helper class reference the rasterizer:

- **D3D11\_RASTERIZER\_DESC2** : structure holding the rasterizer description, noting the CD3D12\_RASTERIZER\_DESC2 helper class for creating rasterizer descriptions.
- **D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS2** : structure holding the boolean `ROVsSupported`, indicating the support.
- **ID3D11Device::CheckFeatureSupport** : method to access the supported features.

## Related topics

[Direct3D 11.3 Features](#)

[Shader Model 5.1](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# Shader Specified Stencil Reference Value (Direct3D 11 Graphics)

Article • 06/18/2021

Enabling pixel shaders to output the Stencil Reference Value, rather than using the API-specified one, enables a very fine granular control over stencil operations.

The shader specified value replaces the API-specified *Stencil Reference Value* for that invocation, which means the change affects both the stencil test, and when stencil op D3D11\_STENCIL\_OP\_REPLACE (one member of [D3D11\\_STENCIL\\_OP](#)) is used to write the reference value to the stencil buffer.

In earlier versions of D3D11, the Stencil Reference Value can only be specified by the [ID3D11DeviceContext::OMSetDepthStencilState](#) method. This means that this value can only be defined on a per-draw granularity. This D3D11.3 feature enables developers to read and use the Stencil Reference Value (`sv_StencilRef`) that is output from a pixel shader, meaning that it can be specified on a per-pixel or per-sample granularity.

This feature is optional in D3D11.3. To test for its support, check the `PSSpecifiedStencilRefSupported` boolean field of [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#) using [ID3D11Device::CheckFeatureSupport](#)

Here is an example of the use of `sv_StencilRef` in a pixel shader:

syntax

```
uint main2(float4 c : COORD) : SV_StencilRef
{
    return uint(c.x);
}
```

## Related topics

[Direct3D 11.3 Features](#)

[Shader Model 5.1](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# Typed Unordered Access View Loads

Article • 06/18/2021

Unordered Access View (UAV) Typed Load is the ability for a shader to read from a UAV with a specific DXGI\_FORMAT.

- [Overview](#)
- [Supported formats and API calls](#)
- [Using typed UAV loads from HLSL](#)
- [Related topics](#)

## Overview

An unordered access view (UAV) is a view of an unordered access resource (which can include buffers, textures, and texture arrays, though without multi-sampling). A UAV allows temporally unordered read/write access from multiple threads. This means that this resource type can be read/written simultaneously by multiple threads without generating memory conflicts. This simultaneous access is handled through the use of [Atomic Functions](#).

D3D12 and D3D11.3 expands on the list of formats that can be used with typed UAV loads.

## Supported formats and API calls

Previously, the following three formats supported typed UAV loads, and were required for D3D11.0 hardware. They are supported for all D3D11.3 and D3D12 hardware.

- R32\_FLOAT
- R32\_UINT
- R32\_SINT

The following formats are supported as a set on D3D12 or D3D11.3 hardware, so if any one is supported, all are supported.

- R32G32B32A32\_FLOAT
- R32G32B32A32\_UINT
- R32G32B32A32\_SINT
- R16G16B16A16\_FLOAT
- R16G16B16A16\_UINT
- R16G16B16A16\_SINT

- R8G8B8A8\_UNORM
- R8G8B8A8\_UINT
- R8G8B8A8\_SINT
- R16\_FLOAT
- R16\_UINT
- R16\_SINT
- R8\_UNORM
- R8\_UINT
- R8\_SINT

The following formats are optionally and individually supported on D3D12 and D3D11.3 hardware, so a single query would need to be made on each format to test for support.

- R16G16B16A16\_UNORM
- R16G16B16A16\_SNORM
- R32G32\_FLOAT
- R32G32\_UINT
- R32G32\_SINT
- R10G10B10A2\_UNORM
- R10G10B10A2\_UINT
- R11G11B10\_FLOAT
- R8G8B8A8\_SNORM
- R16G16\_FLOAT
- R16G16\_UNORM
- R16G16\_UINT
- R16G16\_SNORM
- R16G16\_SINT
- R8G8\_UNORM
- R8G8\_UINT
- R8G8\_SNORM
- 8G8\_SINT
- R16\_UNORM
- R16\_SNORM
- R8\_SNORM
- A8\_UNORM
- B5G6R5\_UNORM
- B5G5R5A1\_UNORM
- B4G4R4A4\_UNORM

To determine the support for any additional formats, call [ID3D11Device::CheckFeatureSupport](#) with the [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#) structure as the first parameter. The

`TypedUAVLoadAdditionalFormats` bit will be set if the "supported as a set" list above is supported. Make a second call to `CheckFeatureSupport`, using a `D3D11_FEATURE_DATA_FORMAT_SUPPORT2` structure (checking the returned structure against the `D3D12_FORMAT_SUPPORT2_UAV_TYPED_LOAD` member of the `D3D11_FORMAT_SUPPORT2` enum) to determine support in the list of optionally supported formats above, for example:

syntax

```
D3D11_FEATURE_DATA_D3D11_OPTIONS2 FeatureData;
ZeroMemory(&FeatureData, sizeof(FeatureData));
HRESULT hr = pDevice->CheckFeatureSupport(D3D11_FEATURE_D3D11_OPTIONS2,
&FeatureData, sizeof(FeatureData));
if (SUCCEEDED(hr))
{
    // TypedUAVLoadAdditionalFormats contains a Boolean that tells you
    // whether the feature is supported or not
    if (FeatureData.TypedUAVLoadAdditionalFormats)
    {
        // Can assume "all-or-nothing" subset is supported (e.g.
        R32G32B32A32_FLOAT)
        // Can not assume other formats are supported, so we check:
        D3D11_FEATURE_DATA_FORMAT_SUPPORT2 FormatSupport;
        ZeroMemory(&FormatSupport, sizeof(FormatSupport));
        FormatSupport.InFormat = DXGI_FORMAT_R32G32_FLOAT;
        hr = pDevice->CheckFeatureSupport(D3D11_FEATURE_FORMAT_SUPPORT2,
&FormatSupport, sizeof(FormatSupport));
        if (SUCCEEDED(hr) && (FormatSupport.OutFormatSupport2 &
D3D11_FORMAT_SUPPORT2_UAV_TYPED_LOAD) != 0)
        {
            // DXGI_FORMAT_R32G32_FLOAT supports UAV Typed Load!
        }
    }
}
```

## Using typed UAV loads from HLSL

For typed UAVs, the HLSL flag is  
`D3D_SHADER_REQUIRES_TYPED_UAV_LOAD_ADDITIONAL_FORMATS`.

Here is example shader code to process a typed UAV load:

syntax

```
RWTexture2D<float4> uav1;
uint2 coord;
float4 main() : SV_Target
{
```

```
    return uav1.Load(coord);  
}
```

## Related topics

[Direct3D 11.3 Features](#)

[Shader Model 5.1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Unified Memory Architecture

Article • 08/23/2019

Querying for whether Unified Memory Architecture (UMA) is supported can help determine how to handle some resources.

A boolean, set by the driver, can be read from the [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#) structure to determine if the hardware supports UMA.

Applications running on UMA may want to have more resources with CPU access enabled than if it is not available. UMA enables applications to avoid copying resource data around, instead of staying efficient solely for non-UMA graphics adapters.

[Direct3D 11.3 Features](#)

## Related topics

[Direct3D 11.3 Features](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Volume Tiled Resources

Article • 05/24/2021

Volume (3D) textures can be used as tiled resources, noting that tile resolution is three-dimensional.

- [Overview](#)
- [D3D11.3 Tiled Resource APIs](#)
- [Related topics](#)

## Overview

Tiled resources decouple a D3D Resource object from its backing memory (resources in the past had a 1:1 relationship with their backing memory). This allows for a variety of interesting scenarios such as streaming in texture data and reusing or reducing memory usage

2D texture tiled resources are supported in D3D11.2. D3D12 and D3D11.3 add support for 3D tiled textures.

The typical resource dimensions used in tiling are 4 x 4 tiles for 2D textures, and 4 x 4 x 4 tiles for 3D textures.

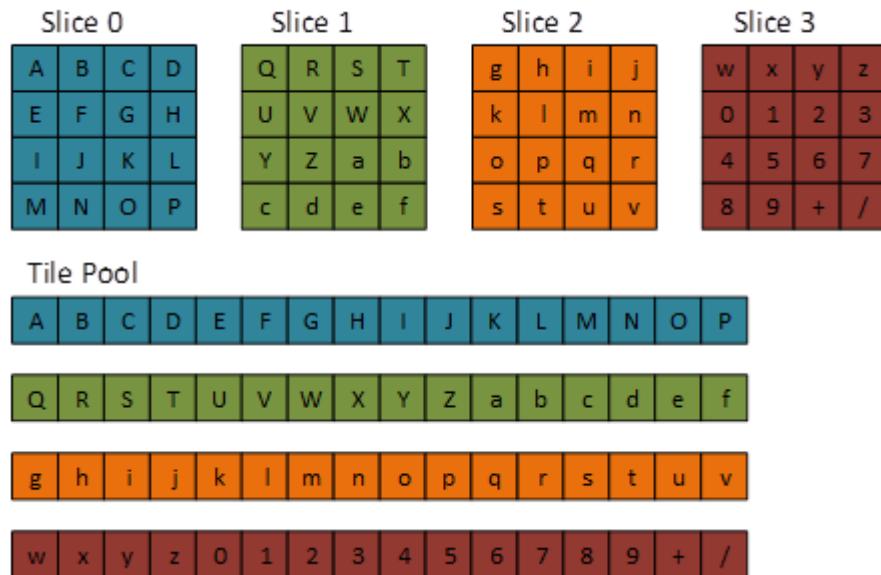
Bits/pixel (1 sample/pixel)	Tile dimensions (pixels, w x h x d)
8	64x32x32
16	32x32x32
32	32x32x16
64	32x16x16
128	16x16x16
BC 1,4	128x64x16
BC 2,3,5,6,7	64x64x16

Note the following formats are not supported with tiled resources: 96bpp formats, video formats, R1\_UNORM, R8G8\_B8G8\_UNORM, R8R8\_G8B8\_UNORM.

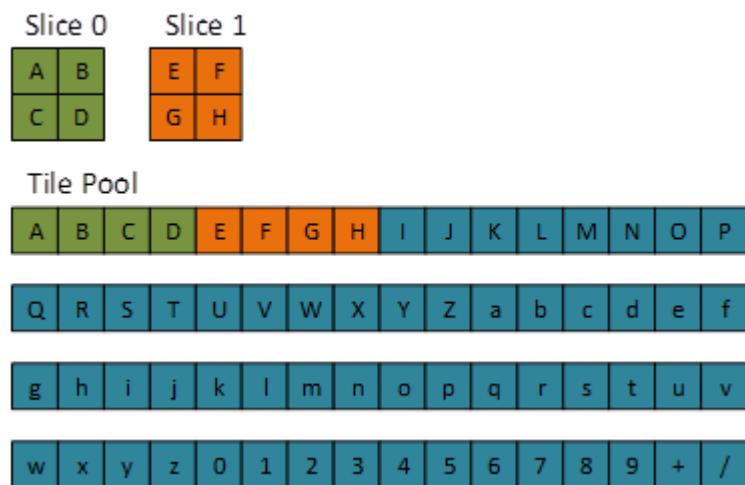
In the diagrams below dark gray represents NULL tiles.

- Texture 3D Tiled Resource default mapping (most detailed mip)
- Texture 3D Tiled Resource default mapping (second most detailed mip)
- Texture 3D Tiled Resource (most detailed mip)
- Texture 3D Tiled Resource (second most detailed mip)
- Texture 3D Tiled Resource (Single Tile)
- Texture 3D Tiled Resource (Uniform Box)

## Texture 3D Tiled Resource default mapping (most detailed mip)



## Texture 3D Tiled Resource default mapping (second most detailed mip)

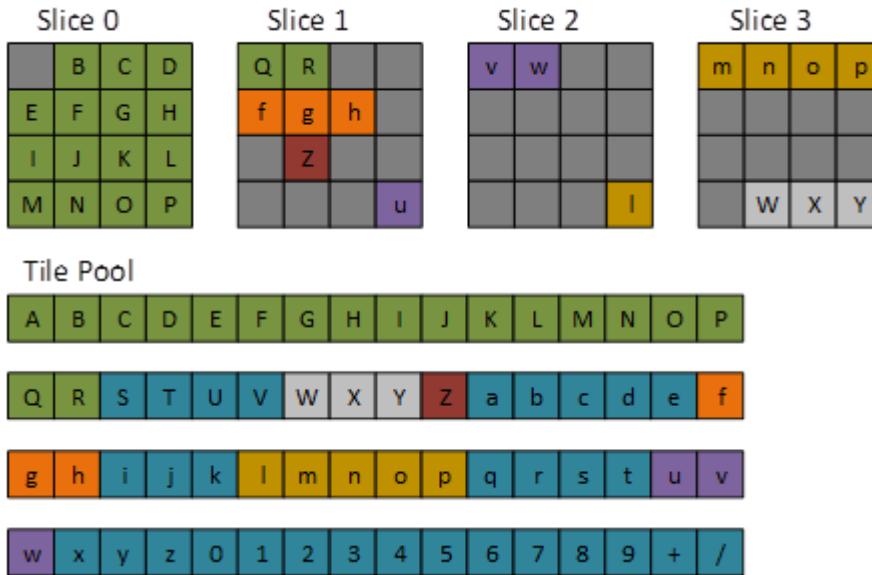


## Texture 3D Tiled Resource (most detailed mip)

The following code sets up a 3D tiled resource at the most detailed mip.

#### Syntax

```
D3D11_TILED_RESOURCE_COORDINATE trCoord;  
trCoord.X = 1;  
trCoord.Y = 0;  
trCoord.Z = 0;  
trCoord.Subresource = 0;  
  
D3D11_TILE_REGION_SIZE trSize;  
trSize.bUseBox = false;  
trSize.NumTiles = 63;
```

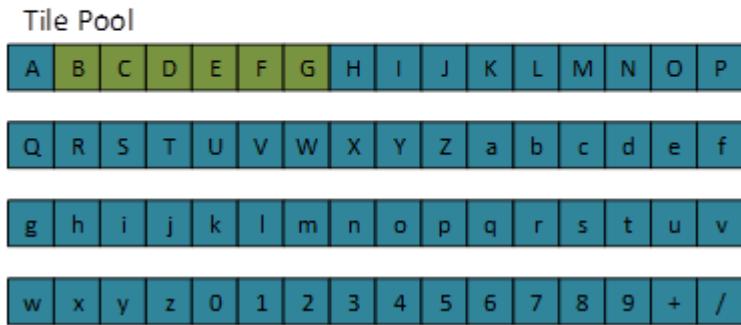
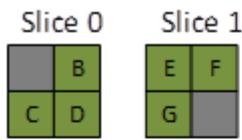


## Texture 3D Tiled Resource (second most detailed mip)

The following code sets up a 3D tiled resource, and the second most detailed mip:

#### Syntax

```
D3D11_TILED_RESOURCE_COORDINATE trCoord;  
trCoord.X = 1;  
trCoord.Y = 0;  
trCoord.Z = 0;  
trCoord.Subresource = 1;  
  
D3D11_TILE_REGION_SIZE trSize;  
trSize.bUseBox = false;  
trSize.NumTiles = 6;
```



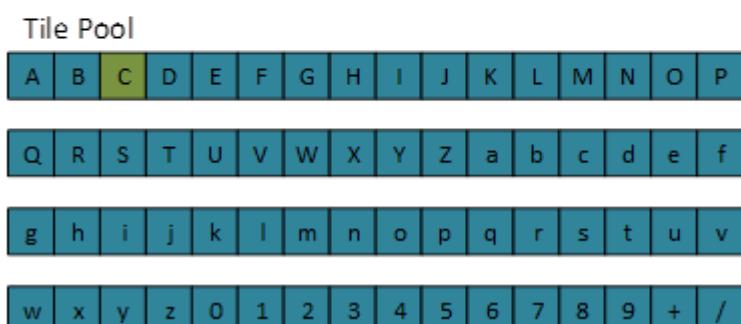
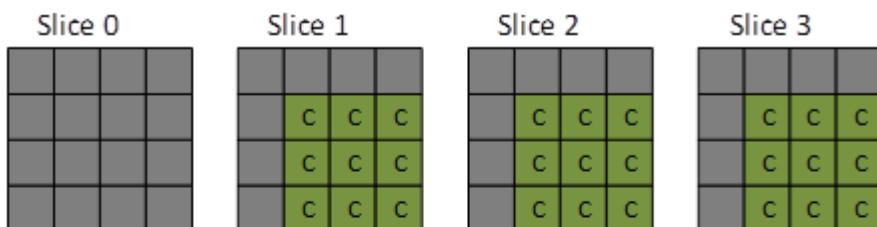
## Texture 3D Tiled Resource (Single Tile)

The following code sets up a Single Tile resource:

syntax

```
D3D11_TILED_RESOURCE_COORDINATE trCoord;
trCoord.X = 1;
trCoord.Y = 1;
trCoord.Z = 1;
trCoord.Subresource = 0;

D3D11_TILE_REGION_SIZE trSize;
trSize.bUseBox = true;
trSize.NumTiles = 27;
trSize.Width = 3;
trSize.Height = 3;
trSize.Depth = 3;
```



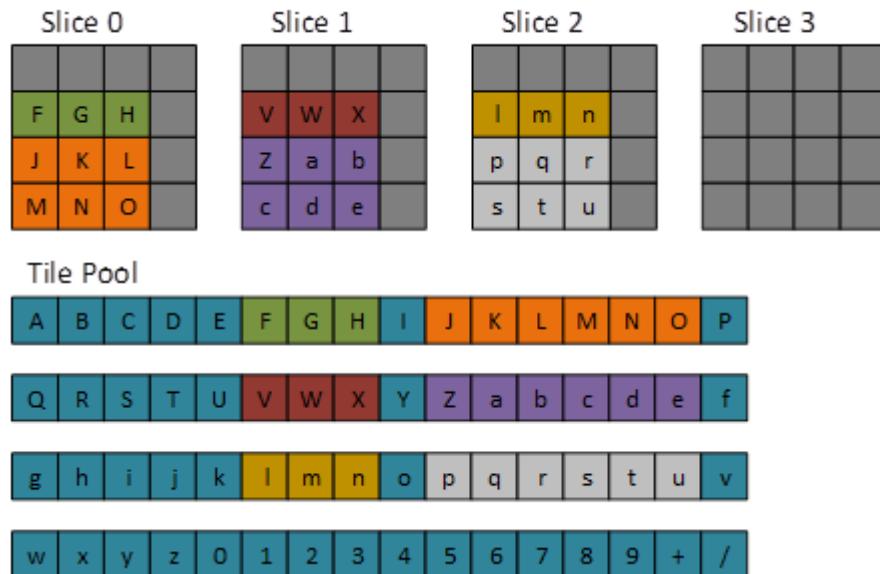
## Texture 3D Tiled Resource (Uniform Box)

The following code sets up a Uniform Box tiled resource (note the statement

```
trSize.bUseBox = true; ) :
```

syntax

```
D3D11_TILED_RESOURCE_COORDINATE trCoord;  
trCoord.X = 0;  
trCoord.Y = 1;  
trCoord.Z = 0;  
trCoord.Subresource = 0;  
  
D3D11_TILE_REGION_SIZE trSize;  
trSize.bUseBox = true;  
trSize.NumTiles = 27;  
trSize.Width = 3;  
trSize.Height = 3;  
trSize.Depth = 3;
```



## D3D11.3 Tiled Resource APIs

The same API calls are used for both 2D and 3D tiled resources:

Enums

- **D3D11\_TILED\_RESOURCES\_TIER** : determines the level of tiled resource support.
- **D3D11\_FORMAT\_SUPPORT2** : used to test for tiled resource support.
- **D3D11\_CHECK\_MULTISAMPLE\_QUALITY\_LEVELS\_FLAG** : determines tiled resource support in a multi-sampling resource.
- **D3D11\_TILE\_COPY\_FLAGS** : holds flags for copying to and from swizzled tiled resources and linear buffers.

Structures

- **D3D11\_TILED\_RESOURCE\_COORDINATE** : holds the x, y, and z co-ordinate, and subresource reference. Note there is a helper class: CD3D11\_TILED\_RESOURCE\_COORDINATE.
- **D3D11\_TILE\_REGION\_SIZE** : specifies the size, and number of tiles, of the tiled region.
- **D3D11\_TILE\_SHAPE** : the tile shape as a width, height and depth in texels.
- **D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS1**: holds the supported tile resource tier level.

## Methods

- **ID3D11Device::CheckFeatureSupport** : used to determine what features, and at what tier, are supported by the current hardware.
- **ID3D11DeviceContext2::CopyTiles** : copies tiles from buffer to tiled resource or vice versa.
- **ID3D11DeviceContext2::UpdateTileMappings** : updates mappings of tile locations in tiled resources to memory locations in a tile pool.
- **ID3D11DeviceContext2::CopyTileMappings** : copies mappings from a source tiled resource to a destination tiled resource.
- **ID3D11DeviceContext2::GetResourceTiling** : gets info about how a tiled resource is broken into tiles.

## Related topics

[Direct3D 11.3 Features](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11.4 Features

Article • 08/19/2020

The following functionality has been added in Direct3D 11.4.

Also see [Where is the DirectX SDK?](#).

## Direct3D device removal

The [RegisterDeviceRemovedEvent](#), and [UnregisterDeviceRemoved](#) methods are supported by a new interface, [ID3D11Device4](#), to support receiving an asynchronous event notification when a Direct3D device has been removed.

## Multithreaded protection

To ensure that graphics commands in particular are executed in a specific order, the [ID3D11Multithread](#) interface has methods to turn multithread protection on and off, and methods to enter and leave critical code requiring this protection.

## Fences for multi-device synchronization and interop with Direct3D 12

The [ID3D11Fence](#), [ID3D11Device5](#) and [ID3D11DeviceContext4](#) provide the same fence functionality as Direct3D 12 for Direct3D 11. Fences are used to synchronize multiple Direct3D11 devices, and for interop between Direct3D 11 and Direct3D 12. Fences are supported in the Windows 10 Creators Update.

## Extended NV12 texture support

NV12 textures with capture and video encode capabilities now support sharing. Older D3D11 texture flags for video encode and capture are deprecated for NV12, as it will be set all the time for new drivers. Such textures can be shared not only with D3D11, but also with D3D12. In D3D12, no new flags represent these texture capabilities.

Refer to the boolean setting in [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS4](#).

## Shader Caching

Drivers may support OS-managed shader caching of Direct3D11 applications in the Windows 10 Creators update.

## Related topics

[What's new in Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Features Introduced In Previous Releases

Article • 01/06/2021

Discover what new features have been added to the previous SDK updates:

## In this section

Topic	Description
<a href="#">What's New in the August 2009 Windows 7/Direct3D 11 SDK</a>	This version of Windows 7/Direct3D 11 ships as part of the DirectX SDK and contains new features, tools, and documentation.
<a href="#">What's New in the November 2008 Direct3D 11 Technical Preview</a>	This version of Direct3D 11 contains new features, tools, and documentation.

## Related topics

[What's new in Direct3D 11](#)

[Direct3D 11 Graphics](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# What's New in the August 2009 Windows 7/Direct3D 11 SDK

Article • 07/09/2024

This version of Windows 7/Direct3D 11 ships as part of the DirectX SDK and contains new features, tools, and documentation.

  Expand table

Item	Description
Direct2D	<p>Direct2D is a hardware-accelerated, immediate-mode, 2-D graphics API that provides high performance and high quality rendering for 2-D geometry, bitmaps, and text. The Direct2D API is designed to interoperate well with Direct3D and GDI. This SDK allows developers to evaluate the API and write simple applications, with some of the more advanced functionality possible on properly configured machines.</p> <p>See <a href="#">Documentation</a> and <a href="#">samples</a>.</p>
DirectWrite	<p>DirectWrite provides support for high-quality text rendering, resolution-independent outline fonts, full Unicode text and layout support, and much, much more:</p> <ul style="list-style-type: none"><li>• A device-independent text layout system that improves text readability in documents and in UI.</li><li>• High-quality, sub-pixel, ClearType text rendering that can use GDI Direct3D, Direct2D, or application-specific rendering technology.</li><li>• Support for multi-format text.</li><li>• Support for the advanced typography features of OpenType fonts.</li><li>• Support for the layout and rendering of text in all languages supported by Windows.</li></ul> <p>This SDK allows developers to evaluate the API and write basic applications for demonstration purposes only.</p> <p>See <a href="#">Documentation</a> and <a href="#">samples</a>.</p>
DXGI 1.1	<p><a href="#">DXGI 1.1</a> builds on DXGI 1.0 and will be available on both Windows Vista and Windows 7. DXGI 1.1 adds several new features:</p> <ul style="list-style-type: none"><li>• Synchronized Shared Surfaces Support. This enables efficient read and write surface sharing between multiple D3D (could be between D3D10 and D3D11) devices.</li><li>• BGRA format support. This allows GDI to render to the same DXGI surface targeted by a Direct2D, Direct3D 10.1 or Direct3D 11 device.</li><li>• Maximum Frame Latency. Using <a href="#">IDXGIDevice1::SetMaximumFrameLatency</a> and <a href="#">IDXGIDevice1::GetMaximumFrameLatency</a>, titles can control the number of frames that are allowed to be stored in a queue, before submission for</li></ul>

Item	Description
	<p>rendering. Latency is often used to control how the CPU chooses between responding to user input and frames that are in the render queue.</p> <ul style="list-style-type: none"><li>• Adapter Enumeration. Using <a href="#">IDXGIFactory1::EnumAdapters1</a>, titles can enumerate local adapters without any monitors or outputs attached, as well as adapters with outputs attached.</li></ul>
Updated Samples	<p>This release has several new and updated samples.</p> <ul style="list-style-type: none"><li>• The new <a href="#">AdaptiveTessellationCS40</a> is an illustration of more advanced compute shader processing techniques that can be run on a D3D10 or D3D11 GPU.</li><li>• The <a href="#">HDR Tone Mapping CS11 sample</a> has been expanded to implement blur and bloom effects (in addition to tone mapping) using compute shader, as well as providing pixel shader implementations for comparison.</li><li>• The <a href="#">Multithreaded Rendering 11 sample</a> has been significantly updated, with more complex art assets and more intensive per-thread processing.</li><li>• The <a href="#">SubD11 sample</a> has been updated with a new facial model, and the sample now leverages the adjacency computation feature of the Samples Content Exporter.</li></ul>

## Related topics

[Features Introduced In Previous Releases](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# What's new in the Nov 2008 Direct3D 11 Tech Preview

Article • 08/19/2021

This version of Direct3D 11 contains new features, tools, and documentation.

Item	Description
Tessellation	Direct3D 11 provides additional pipeline stages to support <a href="#">real-time tessellation</a> of high order primitives. With extensively programmable capabilities, this feature allows many different methods for evaluating high-order surfaces, including subdivision surfaces using approximation techniques, Bezier patches, adaptive tessellation, and displacement mapping. This feature will only be available on Direct3D 11-class hardware, so in order to evaluate this feature you will need to use the Reference Rasterizer. For a demo of tessellation in action, check out the <a href="#">SubD11</a> sample available through the Sample Browser.
Compute Shader	The <a href="#">Compute Shader</a> is an additional stage independent of the Direct3D 11 pipeline that enables general purpose computing on the GPU. In addition to all shader features provided by the unified shader core, the Compute Shader also supports scattered reads and writes to resources through Unordered Access Views, a shared memory pool within a group of executing threads, synchronization primitives, atomic operators, and many other advanced data-parallel features. A variant of the Direct3D 11 Compute Shader has been enabled in this release that can operate on Direct3D 10-class hardware. It is therefore possible to develop Compute Shaders on actual hardware, but an updated driver is required. The full functionality of the Direct3D 11 Compute Shader is intended for support of Direct3D 11-class hardware, so in order to evaluate the full functionality you will need to use the Reference Rasterizer until such hardware is available. For a demo of the Compute Shader in action, check out the <a href="#">HDToneMappingCS11</a> sample available through the Sample Browser.
Multithreaded Rendering	The key API difference from Direct3D 10 in Direct3D 11 is the addition of <a href="#">deferred contexts</a> , which enables scalable execution of Direct3D commands distributed over multiple cores. A Deferred Context captures and assembles actions like state changes and draw submissions that can be executed on the actual device at a later time. By utilizing Deferred Contexts on multiple threads, an application can distribute the CPU overhead needed in the Direct3D11 runtime and the driver to multiple cores, enabling better use of an end-user's machine configuration. This feature is available for use on current Direct3D 10 hardware as well as the reference rasterizer. For a demonstration of API usage, check out the <a href="#">MultithreadedRendering11</a> sample available through the Sample Browser.

Item	Description
Dynamic Shader Linkage	In order to address the combinatorial explosion problem seen in specializing shaders for performance, Direct3D 11 introduces a limited form of runtime <a href="#">shader linkage</a> that allows for near-optimal shader specialization during execution of an application. This is achieved by specifying the implementations of specific functions in shader code when the shader is assigned to the pipeline, allowing the driver to inline native shader instructions quickly rather than forcing the driver to recompile the intermediate language into native instructions with the new configuration. Shader development is exposed through the introduction of classes and interfaces to HLSL. For a demonstration, check out the <a href="#">Dynamic Shader Linkage 11</a> sample available through the Sample Browser.
Windows Advanced Rasterizer (WARP)	Available in this SDK through Direct3D 11 and eventually also through Direct3D 10.1, WARP is a fast, multi-core scaling rasterizer that is fully Direct3D 10.1 compliant. Utilizing this technology is as simple as passing the D3D_DRIVER_TYPE_WARP flag in your device creation.
Direct3D 10 and Direct3D 11 on Direct3D 9 Hardware (D3D10 Level 9)	Available in this SDK through Direct3D 11 and eventually also through Direct3D 10.1, the Direct3D API can target most Direct3D 9 hardware as well as Direct3D 10, Direct3D 10.1 and Direct3D 11 hardware. This is achieved by providing the Feature Level mechanism, which groups hardware into six categories depending on functionality: 9_1, 9_2, 9_3, 10_0, 10_1 and 11_0. A card only meets a feature level if it is fully compliant to that level, and each level is a strict super-set of those below it. Functionality is minimally emulated to assure no unexpected performance cliffs are encountered. Thus, features like Geometry Shaders are not available for Direct3D 9 level targets.
Runtime Binaries	All runtime binaries provided in the Direct3D 11 tech preview that will be available on Windows 7 and Windows Vista SP1 are installed with the SDK and are labeled as "Beta" components (i.e. D3D11_beta.DLL). All beta-labeled components are time-bombed. To create projects to evaluate these new components, you must link to their equivalent beta-labeled import libraries (i.e. D3D11_beta.lib). If you have a PDC copy of Windows 7, the headers, libs, and pdbs provided in the Windows SDK with the build are appropriate for development using the Direct3D 11 components providing in Windows 7. Please reserve the use of the headers, libs, and pdbs in this SDK to the beta components provided herein.
D3DX11	D3DX11 currently supports texture loading, shader compilation, data loading and worker thread functions for Direct3D 11 resources. In the future, this component will provide more of the technologies available in D3DX10. Shader compilation functionality is also provided directly through the Direct3D Compiler component, described next.

Item	Description
HLSL and Direct3D Compiler	<p>The HLSL compiler has several new features for supporting the new technologies available in Direct3D 11. This includes object oriented programming through interfaces and classes, a direct indexing syntax for resource loads, and the 'precise' keyword for ensuring that all operations performed with a specific variable adhere to the strict floating point rules. Almost all new linguistic features have valid functionality on existing shader targets. In addition to supporting all Direct3D 9, Direct3D 10, Direct3D 10.1, and Direct3D 11 shaders the HLSL compiler also supports the special targets needed to write shaders for Direct3D 10 Level 9 targets. The D3D Compiler is now directly accessible outside of D3DX10 and D3DX11 through D3DCompiler.H and D3DCompiler.lib. With these new files, an application is not required to link to D3DX in order to perform runtime compilation, and an application is not required to include the compiler if only D3DX functionality is needed.</p>
D3D11 Reference Rasterizer	<p>The Reference Rasterizer provides a gold-standard rasterization implementation for evaluation of Direct3D 11 features not yet available in hardware. The Reference Rasterizer is also provided as a way to verify a specific hardware implementation's accuracy to the rasterization standard. The reference rasterizer is designed for correctness, not performance. To create a reference device, simply pass the D3D_DRIVER_TYPE_REFERENCE flag at device creation.</p>
D3D11 SDK Layers	<p>Direct3D11 SDK Layers provide a mechanism for tracking the operation of the Direct3D 11 runtime during development. Currently this is used for providing useful debug information, which not only includes errors on improper use but also warnings that recommend best practice use of the runtime and often provides in-depth, useful information for debugging. It is highly recommended that the debug output from D3D11 SDK Layers is turned on at all times during development and an application generates no debug output during execution before it is released or used with PIX for Windows for profiling. Enabling the debug layer is as simple as passing the D3D11_CREATE_DEVICE_DEBUG flag at device creation time. Developers are strongly encouraged to use layers in debug builds. Layers are not recommended for use in profile or release builds.</p>

Item	Description
New Samples	<p>This release has four new samples.</p> <ul style="list-style-type: none"><li>• The <a href="#">Dynamic Shader Linkage 11</a> sample demonstrates use of Shader Model 5 shader interfaces and Direct3D 11 support for linking shader interface methods at runtime.</li><li>• The <a href="#">HDRToneMappingCS11</a> sample demonstrates how to setup and run the Compute Shader(CS for short later on), which is one of the most exciting new features of Direct3D 11. Although the sample only utilizes the CS to implement HDR tone-mapping, the concept should extend easily to other post-processing algorithms as well as more general calculations.</li><li>• The <a href="#">MultithreadedRendering11</a> sample demonstrates how to split rendering among multiple threads, with very low overhead.</li><li>• The new <a href="#">SubD11</a> sample is very similar to the SubD10 sample in the DirectX SDK, except that it has been enhanced to take advantage of three new Direct3D 11 pipeline stages: the hull shader, the tessellator, and the domain shader.</li></ul>

## Related topics

[Features Introduced In Previous Releases](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Programming guide for Direct3D 11

Article • 10/20/2020

The programming guide contains information about how to use the Microsoft Direct3D 11 programmable pipeline to create realtime 3D graphics for games as well as scientific and desktop applications.

## In this section

Topic	Description
<a href="#">Devices</a>	This section describes Direct3D 11 device and device-context objects.
<a href="#">Resources</a>	This section describes aspects of Direct3D 11 resources.
<a href="#">Graphics pipeline</a>	This section describes the Direct3D 11 programmable pipeline.
<a href="#">Compute shader overview</a>	A compute shader is a programmable shader stage that expands Direct3D 11 beyond graphics programming. The compute shader technology is also known as the DirectCompute technology.
<a href="#">Rendering</a>	This section contains information about several Direct3D 11 rendering technologies.
<a href="#">Effects</a>	A DirectX effect is a collection of pipeline state, set by expressions written in <a href="#">HLSL</a> and some syntax that is specific to the effect framework.
<a href="#">Migrating to Direct3D 11</a>	This section provides info for migrating to Direct3D 11 from an earlier version of Direct3D.
<a href="#">Direct3D video interfaces</a>	This topic lists the Direct3D video interfaces that are available in Direct3D 9Ex and that are supported on Windows 7 and later versions of Windows client operating systems and on Windows Server 2008 R2 and later versions of Windows server operating systems.

## Related topics

- [Direct3D 11 Graphics](#)
- [Graphics Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Devices (Direct3D 11 Graphics)

Article • 12/10/2020

A Direct3D device allocates and destroys objects, renders primitives, and communicates with a graphics driver and the hardware. In Direct3D 11, a device is separated into a device object for creating resources and a device-context object, which performs rendering. This section describes Direct3D 11 device and device-context objects.

Objects created from one device cannot be used directly with other devices. Use a shared resource to share data between multiple devices, with the constraint that a shared object can be used only by the device that created it.

## In this section

Topic	Description
<a href="#">Introduction to a Device in Direct3D 11</a>	The Direct3D 11 object model separates resource creation and rendering functionality into a device and one or more contexts; this separation is designed to facilitate multithreading.
<a href="#">Software Layers</a>	The Direct3D 11 runtime is constructed with layers, starting with the basic functionality at the core and building optional and developer-assist functionality in outer layers. This section describes the functionality of each layer.
<a href="#">Limitations Creating WARP and Reference Devices</a>	Some limitations exist for creating WARP and Reference devices in Direct3D 10.1 and Direct3D 11.0. This topic discusses those limitations.
<a href="#">Direct3D 11 on Downlevel Hardware</a>	This section discusses how Direct3D 11 is designed to support both new and existing hardware, from DirectX 9 to DirectX 11.
<a href="#">Using Direct3D 11 feature data to supplement Direct3D feature levels</a>	Find out how to check device support for optional features, including features that were added in recent versions of Windows.

## How to topics about devices

Topic	Description
<a href="#">How To: Create a Reference Device</a>	Describes how to create a reference device.
<a href="#">How To: Create a WARP Device</a>	Describes how to create a WARP device.

Topic	Description
<a href="#">How To: Create a Swap Chain</a>	Describes how to create a swap chain.
<a href="#">How To: Enumerate Adapters</a>	Describes how to enumerate the physical display adapters.
<a href="#">How To: Get Adapter Display Modes</a>	Describes how to get the supported display capabilities of an adapter.
<a href="#">How To: Create a Device and Immediate Context</a>	Describes how to initialize a device.

## Related topics

[Programming Guide for Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Introduction to a Device in Direct3D 11

Article • 04/26/2022

The Direct3D 11 object model separates resource creation and rendering functionality into a device and one or more contexts; this separation is designed to facilitate multithreading.

- [Device](#)
- [Device Context](#)
  - [Immediate Context](#)
  - [Deferred Context](#)
- [Threading Considerations](#)
- [Related topics](#)

## Device

A device is used to create resources and to enumerate the capabilities of a display adapter. In Direct3D 11, a device is represented with an [ID3D11Device](#) interface.

Each application must have at least one device, most applications only create one device. Create a device for one of the hardware drivers installed on your machine by calling [D3D11CreateDevice](#) or [D3D11CreateDeviceAndSwapChain](#) and specify the driver type with the [D3D\\_DRIVER\\_TYPE](#) flag. Each device can use one or more device contexts, depending on the functionality desired.

## Device Context

A device context contains the circumstance or setting in which a device is used. More specifically, a device context is used to set pipeline state and generate rendering commands using the resources owned by a device. Direct3D 11 implements two types of device contexts, one for immediate rendering and the other for deferred rendering; both contexts are represented with an [ID3D11DeviceContext](#) interface.

### Immediate Context

An immediate context renders directly to the driver. Each device has one and only one immediate context which can retrieve data from the GPU. An immediate context can be used to immediately render (or play back) a [command list](#).

There are two ways to get an immediate context:

- By calling either [D3D11CreateDevice](#) or [D3D11CreateDeviceAndSwapChain](#).
- By calling [ID3D11Device::GetImmediateContext](#).

## Deferred Context

A deferred context records GPU commands into a [command list](#). A deferred context is primarily used for multithreading and is not necessary for a single-threaded application. A deferred context is generally used by a worker thread instead of the main rendering thread. When you create a deferred context, it does not inherit any state from the immediate context.

To get a deferred context, call [ID3D11Device::CreateDeferredContext](#).

Any context -- immediate or deferred -- can be used on any thread as long as the context is only used in one thread at a time.

## Threading Considerations

This table highlights the differences in the threading model in Direct3D 11 from prior versions of Direct3D.

Differences between Direct3D 11 and previous versions of Direct3D:

All [ID3D11Device](#) interface methods are free-threaded, which means it is safe to have multiple threads call the functions at the same time.

- All [ID3D11DeviceChild](#)-derived interfaces ([ID3D11Buffer](#), [ID3D11Query](#), etc.) are free-threaded.
- Direct3D 11 splits resource creating and rendering into two interfaces. Map, Unmap, Begin, End, and GetData are implemented on [ID3D11DeviceContext](#) because [ID3D11Device](#) strongly defines the order of operations. [ID3D11Resource](#) and [ID3D11Asynchronous](#) interfaces also implement methods for free-threaded operations.
- The [ID3D11DeviceContext](#) methods (except for those that exist on [ID3D11DeviceChild](#)) are not free-threaded, that is, they require single threading. Only one thread may safely be calling any of its methods (Draw, Copy, Map, etc.) at a time.
- In general, free-threading minimizes the number of synchronization primitives used as well as their duration. However, an application that uses synchronization held for a long time can directly impact how much concurrency an application can expect to achieve.

ID3D10Device interface methods are not designed to be free-threaded. ID3D10Device implements all create and rendering functionality (as does ID3D9Device in Direct3D 9. Map and Unmap are implemented on ID3D10Resource-derived interfaces, Begin, End, and GetData are implemented on ID3D10Asynchronous-derived interfaces.

## Related topics

[Devices](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Software Layers

Article • 08/19/2020

The Direct3D 11 runtime is constructed with layers, starting with the basic functionality at the core and building optional and developer-assist functionality in outer layers. This section describes the functionality of each layer.

As a general rule, layers add functionality, but do not modify existing behavior. For example, core functions will have the same return values independent of the debug layer being instantiated, although additional debug output may be provided if the debug layer is instantiated.

To create layers when a device is created, call [D3D11CreateDevice](#) or [D3D11CreateDeviceAndSwapChain](#) and supply one or more [D3D11\\_CREATE\\_DEVICE\\_FLAG](#) values.

Direct3D 11 provides the following runtime layers:

- [Core Layer](#)
- [Debug Layer](#)
- [Related topics](#)

## Core Layer

The core layer exists by default; providing a very thin mapping between the API and the device driver, minimizing overhead for high-frequency calls. As the core layer is essential for performance, it only performs critical validation. The remaining layers are optional.

## Debug Layer

The debug layer provides extensive additional parameter and consistency validation (such as validating shader linkage and resource binding, validating parameter consistency, and reporting error descriptions).

To create a device that supports the debug layer, you must install the DirectX SDK (to get D3D11SDKLayers.dll), and then specify the [D3D11\\_CREATE\\_DEVICE\\_DEBUG](#) flag when calling the [D3D11CreateDevice](#) function or the [D3D11CreateDeviceAndSwapChain](#) function. If you run your application with the debug layer enabled, the application will be substantially slower. But, to ensure that your application is clean of errors and warnings before you ship it, use the debug layer. For more info, see [Using the debug layer to debug apps](#).

### Note

For Windows 7 with Platform Update for Windows 7 (KB2670838) or Windows 8.x, to create a device that supports the debug layer, install the Windows Software Development Kit (SDK) for Windows 8.x to get D3D11\_1SDKLayers.dll.

### Note

For Windows 10, to create a device that supports the debug layer, enable the "Graphics Tools" optional feature. Go to the Settings panel, under System, Apps & features, Manage optional Features, Add a feature, and then look for "Graphics Tools".

### Note

For info about how to debug DirectX apps remotely, see [Debugging DirectX apps remotely](#).

Alternately, you can enable/disable the debug flag using the [DirectX Control Panel](#) included as part of the DirectX SDK.

When the debug layer lists memory leaks, it outputs a list of object interface pointers along with their friendly names. The default friendly name is "<unnamed>". You can set the friendly name by using the [ID3D11DeviceChild::SetPrivateData](#) method and the [WKPID\\_D3DDDebugObjectName](#) GUID that is in D3Dcommon.h. For example, to name pTexture with a SDKLayer name of mytexture.jpg, use the following code:

```
const char c_szName[] = "mytexture.jpg";
pTexture->SetPrivateData( WKPID_D3DDDebugObjectName, sizeof( c_szName ) - 1,
c_szName );
```

Typically, you should compile these calls out of your production version.

## Related topics

[Devices](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Using the debug layer to debug apps

Article • 08/23/2019

We recommend that you use the [debug layer](#) to debug your apps to ensure that they are clean of errors and warnings. The debug layer helps you write Direct3D code. In addition, your productivity can increase when you use the debug layer because you can immediately see the causes of obscure rendering errors or even black screens at their source. The debug layer provides warnings for many issues. For example, the debug layer provides warnings for these issues:

- Forgot to set a texture but read from it in your pixel shader
- Output depth but have no depth-stencil state bound
- Texture creation failed with INVALIDARG

Here we talk about how to enable the [debug layer](#) and some of the issues that you can prevent by using the debug layer.

- [Enabling the debug layer](#)
- [Preventing errors in your app with the debug layer](#)
  - [Don't pass NULL pointers to Map](#)
  - [Confine source box within source and destination resources](#)
  - [Don't drop DiscardResource or DiscardView](#)
- [Related topics](#)

## Enabling the debug layer

To enable the [debug layer](#), specify the `D3D11_CREATE_DEVICE_DEBUG` flag in the *Flags* parameter when you call the `D3D11CreateDevice` function to create the rendering device. This example code shows how to enable the debug layer when your Microsoft Visual Studio project is in a debug build:

C++

```
UINT creationFlags = D3D11_CREATE_DEVICE_BGRA_SUPPORT;
#if defined(_DEBUG)
    // If the project is in a debug build, enable the debug layer.
    creationFlags |= D3D11_CREATE_DEVICE_DEBUG;
#endif
    // Define the ordering of feature levels that Direct3D attempts to
create.
    D3D_FEATURE_LEVEL featureLevels[] =
{
    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_1
    };

    ComPtr<ID3D11Device> d3dDevice;
    ComPtr<ID3D11DeviceContext> d3dDeviceContext;
    DX::ThrowIfFailed(
        D3D11CreateDevice(
            nullptr,                         // specify nullptr to use the
        default adapter
            D3D_DRIVER_TYPE_HARDWARE,
            nullptr,                         // specify nullptr because
        D3D_DRIVER_TYPE_HARDWARE
                                         // indicates that this function
uses hardware
            creationFlags,                  // optionally set debug and
        Direct2D compatibility flags
            featureLevels,
            ARRSIZE(featureLevels),
            D3D11_SDK_VERSION,              // always set this to
        D3D11_SDK_VERSION
            &d3dDevice,
            nullptr,
            &d3dDeviceContext
        )
    );

```

## Preventing errors in your app with the debug layer

If you misuse the Direct3D 11 API or pass bad parameters, the debug output of the [debug layer](#) reports an error or a warning. You can then correct your mistake. Next, we look at some coding issues that can cause undefined behavior or even the operating system to crash. You can catch and prevent these issues by using the debug layer.

### Don't pass NULL pointers to Map

If you pass **NULL** to the *pResource* or *pMappedResource* parameter of the [ID3D11DeviceContext::Map](#) method, the behavior of **Map** is undefined. If you created a device that just supports the [core layer](#), invalid parameters to **Map** can crash the operating system. If you created a device that supports the [debug layer](#), the debug output reports an error on this invalid **Map** call.

## Confine source box within source and destination resources

In a call to the [ID3D11DeviceContext::CopySubresourceRegion](#) method, the source box must be within the source resource. The destination offsets, (x, y, and z) allow the source box to be offset when writing into the destination resource, but the dimensions of the source box and the offsets must be within the size of the resource. If you try to copy outside the destination resource or specify a source box that is larger than the source resource, the behavior of **CopySubresourceRegion** is undefined. If you created a device that supports the [debug layer](#), the debug output reports an error on this invalid **CopySubresourceRegion** call. Invalid parameters to **CopySubresourceRegion** cause undefined behavior and might result in incorrect rendering, clipping, no copy, or even the removal of the rendering device.

## Don't drop DiscardResource or DiscardView

The runtime drops a call to [ID3D11DeviceContext1::DiscardResource](#) or [ID3D11DeviceContext1::DiscardView](#) unless you correctly create the resource.

The resource that you pass to [ID3D11DeviceContext1::DiscardResource](#) must have been created by using [D3D11\\_USAGE\\_DEFAULT](#) or [D3D11\\_USAGE\\_DYNAMIC](#), otherwise the runtime drops the call to **DiscardResource**.

The resource that underlies the view that you pass to [ID3D11DeviceContext1::DiscardView](#) must have been created using [D3D11\\_USAGE\\_DEFAULT](#) or [D3D11\\_USAGE\\_DYNAMIC](#), otherwise the runtime drops the call to **DiscardView**.

If you created a device that supports the [debug layer](#), the debug output reports an error regarding the dropped call.

## Related topics

[Software Layers](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Limitations Creating WARP and Reference Devices

Article • 08/19/2020

Some limitations exist for creating WARP and Reference devices in Direct3D 10.1 and Direct3D 11.0. This topic discusses those limitations.

The D3D10\_DRIVER\_TYPE\_WARP and D3D10\_DRIVER\_TYPE\_REFERENCE driver types are not supported on the D3D10\_FEATURE\_LEVEL\_9\_1 through D3D10\_FEATURE\_LEVEL\_9\_3 feature levels in Direct3D 10.1. Additionally, the D3D\_DRIVER\_TYPE\_WARP driver type is not supported on D3D\_FEATURE\_LEVEL\_11\_0 in Direct3D 11.0. That is, when you call [D3D10CreateDevice1](#) to create a Direct3D 10.1 device or when you call [D3D11CreateDevice](#) to create a Direct3D 11.0 device, if you specify a combination of one of these driver types with one of these feature levels in the call, the call is invalid. Only the following combinations of feature levels, runtimes, and driver types are valid for WARP and Reference devices:

- D3D\_DRIVER\_TYPE\_WARP on all feature levels in Direct3D 11.1, which Windows 8 includes

D3D\_DRIVER\_TYPE\_REFERENCE on all feature levels in Direct3D 11.1

When you call [D3D11CreateDevice](#) to create a Direct3D 11.1 device, the call is valid if you specify a combination of one of these driver types with one of these feature levels.

- D3D\_DRIVER\_TYPE\_WARP on D3D\_FEATURE\_LEVEL\_9\_1 through D3D\_FEATURE\_LEVEL\_10\_1 feature levels in Direct3D 11

D3D\_DRIVER\_TYPE\_REFERENCE on D3D\_FEATURE\_LEVEL\_9\_1 through D3D\_FEATURE\_LEVEL\_11\_0 feature levels in Direct3D 11

When you call [D3D11CreateDevice](#) to create a Direct3D 11 device, the call is valid if you specify a combination of one of these driver types with one of these feature levels.

- D3D10\_DRIVER\_TYPE\_WARP and D3D10\_DRIVER\_TYPE\_REFERENCE on D3D10\_FEATURE\_LEVEL\_10\_0 through D3D10\_FEATURE\_LEVEL\_10\_1 feature levels in Direct3D 10.1

When you call [D3D10CreateDevice1](#) to create a Direct3D 10.1 device, the call is valid if you specify a combination of one of these driver types with one of these

feature levels.

## Related topics

[Devices](#)

[Introduction to Direct3D 11 on Downlevel Hardware](#)

[How To: Create a WARP Device](#)

[How To: Create a Reference Device](#)

[D3D10\\_DRIVER\\_TYPE](#)

[D3D10\\_FEATURE\\_LEVEL1](#)

[D3D\\_DRIVER\\_TYPE](#)

[D3D\\_FEATURE\\_LEVEL](#)

---

## Feedback

Was this page helpful?

 Yes

 No

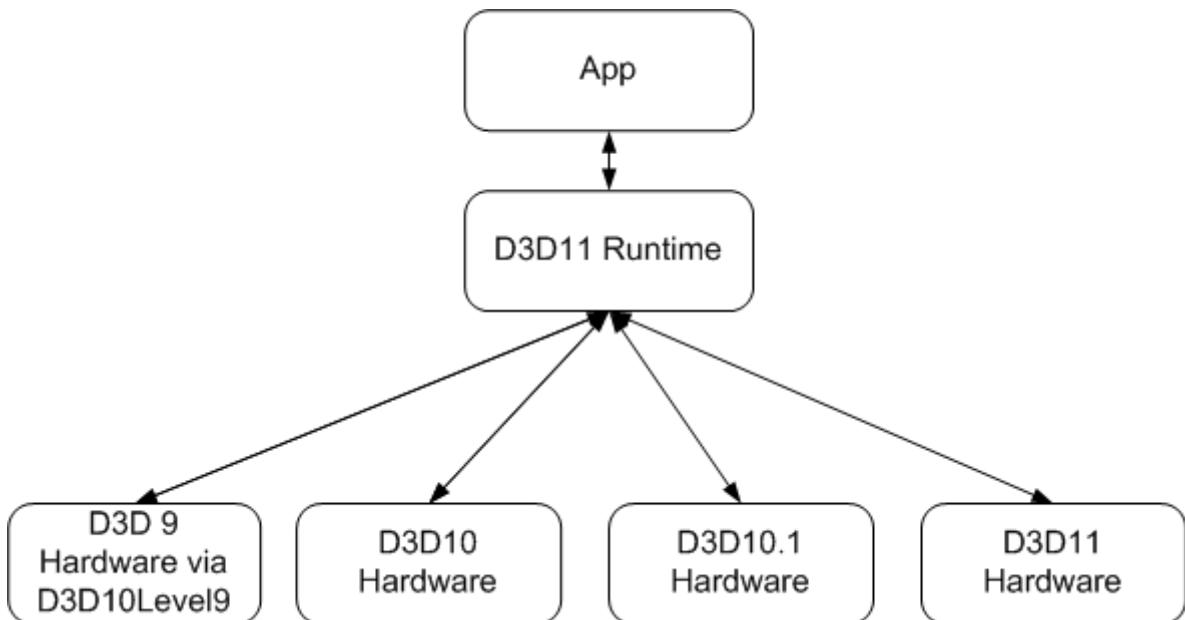
[Get help at Microsoft Q&A](#)

# Direct3D 11 on Downlevel Hardware

Article • 11/04/2020

This section discusses how Direct3D 11 is designed to support both new and existing hardware, from DirectX 9 to DirectX 11.

This diagram shows how Direct3D 11 supports new and existing hardware.



With Direct3D 11, a new paradigm is introduced called feature levels. A feature level is a well defined set of GPU functionality. Using a feature level, you can target a Direct3D application to run on a downlevel version of Direct3D hardware.

The [10Level9 Reference](#) section lists the differences between how various [\*\*ID3D11Device\*\*](#) and [\*\*ID3D11DeviceContext\*\*](#) methods behave at various 10Level9 feature levels.

## In this section

Topic	Description
<a href="#">Direct3D feature levels</a>	This topic discusses Direct3D feature levels.
<a href="#">Exceptions</a>	This topic describes exceptions when using Direct3D 11 on downlevel hardware.
<a href="#">Compute Shaders on Downlevel Hardware</a>	This topic discusses how to make use of <a href="#">compute shaders</a> in a Direct3D 11 app on Direct3D 10 hardware.

Topic	Description
Preventing Unwanted NULL Pixel Shader SRVs	This topic discusses how to work around the driver receiving <b>NULL</b> shader-resource views (SRVs) even when non- <b>NULL</b> SRVs are bound to the pixel shader stage.

## How to topics about feature levels

Topic	Description
<a href="#">How To: Get the Device Feature Level</a>	How to get a feature level.

## Related topics

[Devices](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D feature levels

Article • 07/24/2023

To handle the diversity of video cards in new and existing machines, Microsoft Direct3D 11 introduces the concept of feature levels. This topic discusses Direct3D feature levels.

Each video card implements a certain level of Microsoft DirectX (DX) functionality depending on the graphics processing units (GPUs) installed. In prior versions of Microsoft Direct3D, you could find out the version of Direct3D the video card implemented, and then program your application accordingly.

With Direct3D 11, a new paradigm is introduced called feature levels. A feature level is a well-defined set of GPU functionality. For instance, the 9\_1 feature level implements the functionality that was implemented in Microsoft Direct3D 9, which exposes the capabilities of shader models [ps\\_2\\_x](#) and [vs\\_2\\_x](#), while the 11\_0 feature level implements the functionality that was implemented in Direct3D 11.

Now when you create a device, you can attempt to create a device for the feature level that you want to request. If the device creation works, that feature level exists, if not, the hardware does not support that feature level. You can either try to recreate a device at a lower feature level or you can choose to exit the application. For more info about creating a device, see the [D3D11CreateDevice](#) function.

Using feature levels, you can develop an application for Direct3D 9, Microsoft Direct3D 10, or Direct3D 11, and then run it on 9, 10 or 11 hardware (with some exceptions; for example, new 11 features will not run on an existing 9 card). Here is a couple of other basic properties of feature levels:

- A GPU that allows a device to be created meets or exceeds the functionality of that feature level.
- A feature level always includes the functionality of previous or lower feature levels.
- A feature level does not imply performance, only functionality. Performance is dependent on hardware implementation.
- Choose a feature level when you create a Direct3D 11 device.

For information about limitations creating nonhardware-type devices on certain feature levels, see [Limitations Creating WARP and Reference Devices](#).

To assist you in deciding what feature level to design with, compare the features for each feature level.

The [10Level9 Reference](#) section lists the differences between how various [ID3D11Device](#) and [ID3D11DeviceContext](#) methods behave at various 10Level9 feature levels.

## Formats of version numbers

There are three formats for Direct3D versions, shader models, and feature levels.

- Direct3D versions use a period; for example, Direct3D 12.0.
- Shader models use a period; for example, shader model 5.1.
- Feature levels use an underscore; for example, feature level 12\_0.

## Direct3D 12 feature support (feature levels 12\_2 through 11\_0)

The following features are available for the feature levels listed. The headings across the top row are Direct3D 12 feature levels. The headings in the left-hand column are features. Also see [Footnotes for the tables](#).

Feature \ Feature Level	12_2 <sup>8</sup>	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0
Shader model	6.5	5.1 <sup>2</sup>	5.1 <sup>2</sup>	5.1 <sup>2</sup>	5.1 <sup>2</sup>
WDDM driver model	2.0	1.x	1.x	1.x	1.x
Raytracing	Tier 1.1	Optional	Optional	Optional	Optional
Variable shading rate	Tier 2	Optional	Optional	Optional	Optional
Mesh shader	Tier 1	Optional	Optional	Optional	Optional
Sampler feedback	Tier 0.9	Optional	Optional	Optional	Optional

Feature \ Feature Level	12_2 <sup>8</sup>	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0
Resource binding	Tier 3	Tier 3	Tier 3	Tier 3	Tier 1
Root signature	1.1	1	1	1	1
Depth bounds test	Yes	Optional	Optional	Optional	Optional
Write buffer immediate	Direct, Compute, Bundle	Optional	Optional	Optional	Optional
GPU virtual address bits	40 <sup>10</sup>	40 <sup>10</sup>	40 <sup>10</sup>		
<b>Tiled resources</b>	Tier3	Tier2 <sup>6</sup>	Tier2 <sup>6</sup>	Optional	Optional
<b>Conservative Rasterization</b>	Tier3	Tier1 <sup>6</sup>	Optional	Optional	No
<b>Rasterizer Order Views</b>	Yes	Yes	Optional	Optional	No
<b>Min/Max Filters</b>	Yes	Yes	Yes	Optional	No
Map Default Buffer	N/A	Optional	Optional	Optional	Optional
<b>Shader Specified Stencil Reference Value</b>	Optional	Optional	Optional	Optional	No
Typed Unordered Access View Loads	18 formats, more optional	18 formats, more optional	18 formats, more optional	3 formats, more optional	3 formats, more optional
<b>Geometry Shader</b>	Yes	Yes	Yes	Yes	Yes
<b>Stream Out</b>	Yes	Yes	Yes	Yes	Yes
<b>DirectCompute / Compute Shader</b>	Yes	Yes	Yes	Yes	Yes
<b>Hull and Domain Shaders</b>	Yes	Yes	Yes	Yes	Yes
Feature \ Feature Level	12_2 <sup>8</sup>	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0
<b>Texture Resource Arrays</b>	Yes	Yes	Yes	Yes	Yes
<b>Cubemap Resource Arrays</b>	Yes	Yes	Yes	Yes	Yes
<b>BC4/BC5 Compression</b>	Yes	Yes	Yes	Yes	Yes
<b>BC6H/BC7 Compression</b>	Yes	Yes	Yes	Yes	Yes
<b>Alpha-to-coverage</b>	Yes	Yes	Yes	Yes	Yes
<b>Extended Formats (BGRA, and so on)</b>	Yes	Yes	Yes	Yes	Yes
<b>10-bit XR High Color Format</b>	Yes	Yes	Yes	Yes	Yes
<b>Logic Operations (Output Merger)</b>	Yes	Yes	Yes	Yes	Optional <sup>1</sup>
Target-independent rasterization	Yes	Yes	Yes	Yes	No
<b>Multiple render target(MRT) with ForcedSampleCount 1</b>	Yes	Yes	Yes	Yes	Optional <sup>1</sup>
UAV slots	Tiered <sup>9</sup>	64	64	64	8
UVAs at every stage	Yes	Yes	Yes	Yes	No
Feature \ Feature Level	12_2 <sup>8</sup>	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0
<b>Max forced sample count for UAV-only rendering</b>	16	16	16	16	8
Constant buffer offsetting and partial updates	Yes	Yes	Yes	Yes	Optional <sup>1</sup>
16 bits per pixel (bpp) formats	Yes	Yes	Yes	Yes	Optional <sup>1</sup>
Max Texture Dimension	16384	16384	16384	16384	16384
Max Cubemap Dimension	16384	16384	16384	16384	16384

Feature \ Feature Level	12_2 <sup>8</sup>	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0
Max Volume Extent	2048	2048	2048	2048	2048
Max Texture Repeat	16384	16384	16384	16384	16384
Max Anisotropy	16	16	16	16	16
Max Primitive Count	2^32 – 1	2^32 – 1	2^32 – 1	2^32 – 1	2^32 – 1
Max Vertex Index	2^32 – 1	2^32 – 1	2^32 – 1	2^32 – 1	2^32 – 1
Max Input Slots	32	32	32	32	32
Simultaneous Render Targets	8	8	8	8	8
Feature \ Feature Level	12_2 <sup>8</sup>	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0
Occlusion Queries	Yes	Yes	Yes	Yes	Yes
Separate Alpha Blend	Yes	Yes	Yes	Yes	Yes
Mirror Once	Yes	Yes	Yes	Yes	Yes
Overlapping Vertex Elements	Yes	Yes	Yes	Yes	Yes
Independent Write Masks	Yes	Yes	Yes	Yes	Yes
Instancing	Yes	Yes	Yes	Yes	Yes
Nonpowers-of-2 conditionally <sup>3</sup>	No	No	No	No	No
Nonpowers-of-2 unconditionally <sup>4</sup>	Yes	Yes	Yes	Yes	Yes

Additionally, the following flags are set:

Feature \ Feature Level	12_2 <sup>8</sup>
WaveOps	TRUE
OutputMergerLogicOp	TRUE
VPAndRTArrayIndexFromAnyShaderFeedingRasterizerSupportWithoutGSEmulation	TRUE
CopyQueueTimestampQueriesSupported	TRUE
CastingFullyTypedFormatSupported	TRUE
Int64ShaderOps	TRUE

## Direct3D 11 feature support (feature levels 11\_1 through 9\_1)

The following features are available for the feature levels listed. The headings across the top row are Direct3D 11 feature levels. The headings in the left-hand column are features. Also see [Footnotes for the tables](#).

Feature \ Feature Level	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0	10_1	10_0	9_3 <sup>7</sup>	9_2	9_1
Shader model	5.1 <sup>2</sup>	5.1 <sup>2</sup>	5.0 <sup>2</sup>	5.0 <sup>2</sup>	4.x	4.0	2.0 (4_0_level_9_3)	2.0 (4_0_level_9_1)	2.0 (4_0_level_9_1) [vs_2_a/ps_2_x] <sup>5</sup>
WDDM driver model	1.x	1.x	1.x	1.x	1.x	1.x	1.x	1.x	1.x
Tiled resources	Tier2 <sup>6</sup>	Tier2 <sup>6</sup>	Optional	Optional	No	No	No	No	No
Conservative Rasterization	Tier1 <sup>6</sup>	Optional	Optional	No	No	No	No	No	No
Rasterizer Order Views	Yes	Optional	Optional	No	No	No	No	No	No

Feature \ Feature Level	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0	10_1	10_0	9_3 <sup>7</sup>	9_2	9_1
Min/Max Filters	Yes	Yes	Optional	No	No	No	No	No	No
Map Default Buffer	Optional	Optional	Optional	Optional	No	No	No	No	No
Shader Specified Stencil Reference Value	Optional	Optional	Optional	No	No	No	No	No	No
Typed Unordered Access View Loads	18 formats, more optional	18 formats, more optional	3 formats, more optional	3 formats, more optional	No	No	No	No	No
Geometry Shader	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Stream Out	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
DirectCompute / Compute Shader	Yes	Yes	Yes	Yes	Optional	Optional	N/A	N/A	N/A
Hull and Domain Shaders	Yes	Yes	Yes	Yes	No	No	No	No	No
Feature \ Feature Level	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0	10_1	10_0	9_3 <sup>7</sup>	9_2	9_1
Texture Resource Arrays	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Cubemap Resource Arrays	Yes	Yes	Yes	Yes	Yes	No	No	No	No
BC4/BC5 Compression	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
BC6H/BC7 Compression	Yes	Yes	Yes	Yes	No	No	No	No	No
Alpha-to-coverage	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Extended Formats (BGRA, and so on)	Yes	Yes	Yes	Yes	Optional	Optional	Yes	Yes	Yes
10-bit XR High Color Format	Yes	Yes	Yes	Yes	Optional	Optional	N/A	N/A	N/A
Logic Operations (Output Merger)	Yes	Yes	Yes	Optional <sup>1</sup>	Optional <sup>1</sup>	Optional <sup>1</sup>	No	No	No
Target-independent rasterization	Yes	Yes	Yes	Yes	Yes	No	No	No	No
Multiple render target(MRT) with ForcedSampleCount 1	Yes	Optional <sup>1</sup>	Optional <sup>1</sup>	Optional <sup>1</sup>	No	No	No		
UAV slots	64	64	64	8	1	1	N/A	N/A	N/A
UVAs at every stage	Yes	Yes	Yes	No	No	No	N/A	N/A	N/A
Feature \ Feature Level	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0	10_1	10_0	9_3 <sup>7</sup>	9_2	9_1
Max forced sample count for UAV-only rendering	16	16	16	8	N/A	N/A	N/A	N/A	N/A
Constant buffer offsetting and partial updates	Yes	Yes	Yes	Optional <sup>1</sup>	Optional <sup>1</sup>	Optional <sup>1</sup>	Yes <sup>1</sup>	Yes <sup>1</sup>	Yes <sup>1</sup>

Feature \ Feature Level	12_1 <sup>0</sup>	12_0 <sup>0</sup>	11_1 <sup>1</sup>	11_0	10_1	10_0	9_3 <sup>7</sup>	9_2	9_1
16 bits per pixel (bpp) formats	Yes	Yes	Yes	Optional <sup>1</sup>					
Max Texture Dimension	16384	16384	16384	16384	8192	8192	4096	2048	2048
Max Cubemap Dimension	16384	16384	16384	16384	8192	8192	4096	512	512
Max Volume Extent	2048	2048	2048	2048	2048	2048	256	256	256
Max Texture Repeat	16384	16384	16384	16384	8192	8192	8192	2048	128
Max Anisotropy	16	16	16	16	16	16	16	16	2
Max Primitive Count	2 <sup>32</sup> – 1	2 <sup>32</sup> – 1	2 <sup>32</sup> – 1	1048575	1048575	65535			
Max Vertex Index	2 <sup>32</sup> – 1	2 <sup>32</sup> – 1	2 <sup>32</sup> – 1	1048575	1048575	65534			
Max Input Slots	32	32	32	32	32	16	16	16	16
Simultaneous Render Targets	8	8	8	8	8	8	4	1	1
Feature \ Feature Level	12_2 <sup>8</sup>	12_1 <sup>0</sup>	11_1 <sup>1</sup>	11_0	10_1	10_0	9_3 <sup>7</sup>	9_2	9_1
Occlusion Queries	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Separate Alpha Blend	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Mirror Once	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Overlapping Vertex Elements	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Independent Write Masks	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Instancing	Yes	Yes	Yes	Yes	Yes	Yes	Yes <sup>7</sup>	No	No
Nonpowers-of-2 conditionally <sup>3</sup>	No	No	No	No	No	No	Yes	Yes	Yes
Nonpowers-of-2 unconditionally <sup>4</sup>	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No

## Footnotes for the tables

<sup>0</sup> Requires the Direct3D 11.3 or Direct3D 12 runtime.

<sup>1</sup> Requires the Direct3D 11.1 runtime.

<sup>2</sup> Shader model 5.0 and above can optionally support double-precision shaders, extended double-precision shaders, the **SAD4** shader instruction, and partial-precision shaders. To determine the shader model 5.0 options that are available for DirectX 11, call [ID3D11Device::CheckFeatureSupport](#). Some compatibility depends on what hardware you are running on. Shader model 5.1 and above are only supported through the DirectX 12 API, regardless of the feature level that's being used. DirectX 11 only supports up to shader model 5.0. The DirectX 12 API only goes down to feature level 11\_0.

<sup>3</sup> At feature levels 9\_1, 9\_2 and 9\_3, the display device supports the use of 2-D textures with dimensions that are not powers of two under two conditions. First, only one MIP-map level for each texture can be created, and second, no wrap sampler modes for textures are allowed (that is, the **AddressU**, **AddressV**, and **AddressW** members of [D3D11\\_SAMPLER\\_DESC](#) cannot be set to [D3D11\\_TEXTURE\\_ADDRESS\\_WRAP](#)).

<sup>4</sup> At feature levels 10\_0, 10\_1 and 11\_0, the display device unconditionally supports the use of 2-D textures with dimensions that are not powers of two.

<sup>5</sup> Vertex Shader 2a with 256 instructions, 32 temporary registers, static flow control of depth 4, dynamic flow control of depth 24, and D3DVS20CAPS\_PREDICTION. Pixel Shader 2x with 512 instructions, 32 temporary registers, static flow control of depth 4, dynamic flow control of depth 24, D3DPS20CAPS\_ARBITRARYSWIZZLE, D3DPS20CAPS\_GRADIENTINSTRUCTIONS, D3DPS20CAPS\_PREDICTION, D3DPS20CAPS\_NODEPENDENTREADLIMIT, and D3DPS20CAPS\_NOTELEXINSTRUCTIONLIMIT.

<sup>6</sup> Higher tiers optional.

<sup>7</sup> For Feature Level 9\_3, the only rendering methods supported are **Draw**, **DrawIndexed**, and **DrawIndexInstanced**. Also for Feature Level 9\_3, point list rendering is supported only for rendering via **Draw**.

<sup>8</sup> Supported by Windows 11.

<sup>9</sup> In the Direct3D 12 API there are limits on the number of descriptors in a CBV/SRV/UAV heap. See [Hardware Tiers](#) for details. Separately, there's a limit on the number of UAVs in all descriptor tables across all stages, which is based on [resource binding tier](#).

<sup>10</sup> A 64-bit process requires 40 bits of address space available per resource and per process. A 32-bit process might be limited to 31 bits of address space. There are two capabilities (caps) available in the API—per-process and per-resource. Per-process address space is always greater than or equal to the per-resource address space.

For details of format support at different hardware feature levels, 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](#)

## Related topics

- [Direct3D 11 on downlevel hardware](#)
- [Hardware feature levels \(Direct3D 12\)](#)

---

## Feedback

Was this page helpful?



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

# Exceptions

Article • 08/19/2020

Some features of Direct3D 11 are not fully specified by feature levels. This topic describes exceptions when using Direct3D 11 on downlevel hardware. Perhaps a feature was added after the feature level is defined (and requires an updated driver) or perhaps different GPUs implement widely different implementations. Feature level exceptions can be gathered into the following groups:

- [Extended Formats](#)
- [Multisample Anti-Aliasing](#)
- [Texture2D Sizes](#)
- [Special Behavior of Adapters for Feature Level 9](#)
- [Related topics](#)

The [10Level9 Reference](#) section lists the differences between how various [ID3D11Device](#) and [ID3D11DeviceContext](#) methods behave at various 10Level9 feature levels.

## Extended Formats

An extended format is a pixel format added to Direct3D 10.1 and Direct3D 11 for feature levels 10\_0 and 10\_1. An extended format requires an updated driver (for Direct3D 10\_1 or below). Use [ID3D11Device::CheckFormatSupport](#) and [ID3D11Device::CheckFeatureSupport](#) to query for support for these extended formats.

An extended format:

- Adds support for BGRA order of 8-bit per-component resources.
- Allows casting of an integer-value swap-chain buffer. This allows an application to add or remove the \_SRGB suffix or render to an XR\_BIAS swap chain.
- Adds optional support for DXGI\_FORMAT\_R10G10B10\_XR\_BIAS\_A2\_UNORM.
- Guarantees that a DXGI\_FORMAT\_R16G16B16A16\_FLOAT swap chain is presented as if the data contained is not sRGB-encoded.

The full set of extended formats are either fully supported or not supported, with the exception of the XR\_BIAS format. The XR\_BIAS format is:

- Unsupported in any 9 level
- Optional in either the 10\_0 or 10\_1 level
- Guaranteed at the 11\_0 level

# Multisample Anti-Aliasing

MSAA implementations have little in common across GPU implementations. Feature Level 10.1 added some well-defined minima, but at lower feature levels, MSAA must be tested explicitly using [ID3D11Device::CheckMultisampleQualityLevels](#).

## Texture2D Sizes

A feature level guarantees that a minimum size can be created, however, an application can create larger textures up to the full size supported by the GPU. An application should expect failure from a method such as [ID3D11Device::CreateTexture2D](#) if a maximum is exceeded.

## Special Behavior of Adapters for Feature Level 9

The three lowest feature levels D3D\_FEATURE\_LEVEL\_9\_1, D3D\_FEATURE\_LEVEL\_9\_2 and D3D\_FEATURE\_LEVEL\_9\_3, share a common implementation DLL and treat the [IDXGIAdapter](#) argument to D3D11CreateDevice[AndSwapchain] as a template adapter and create their own adapter as part of device creation. This means that the [IDXGIAdapter](#) passed into the creation routine will not be the same adapter as that retrieved from the device via [IDXGIDevice::GetAdapter](#). The impact of this is that the [IDXGIOOutputs](#) enumerated from the passed-in adapter cannot be used to enter fullscreen using any level 9 device, since those outputs are not owned by the device's adapter. It is good practice to discard the passed-in template adapter and retrieve the device's created adapter using [IDXGIDevice::GetAdapter](#), where [IDXGIDevice](#) can be retrieved using [QueryInterface](#) from the Direct3D device interface.

## Related topics

[Direct3D 11 on Downlevel Hardware](#)

---

## Feedback



Was this page helpful? [!\[\]\(dc0c48fa9d7638b4af4bf013a0e31768\_img.jpg\) Yes](#) [!\[\]\(2e77a1152955a1bc1706b38ba55f5ee6\_img.jpg\) No](#)

Get help at Microsoft Q&A

# Compute Shaders on Downlevel Hardware

Article • 10/20/2020

Direct3D 11 provides the ability to use [compute shaders](#) that operate on most Direct3D 10.x hardware, with some limitations to operation. The compute shader technology is also known as the DirectCompute technology. This topic discusses how to make use of [compute shaders](#) in a Direct3D 11 app on Direct3D 10 hardware.

Support for compute shaders on downlevel hardware is only for devices compatible with Direct3D 10.x. Compute shaders cannot be used on Direct3D 9.x hardware.

To check if Direct3D 10.x hardware supports compute shaders, call [ID3D11Device::CheckFeatureSupport](#). In the `CheckFeatureSupport` call, pass the `D3D11_FEATURE_D3D10_X_HARDWARE_OPTIONS` value to the `Feature` parameter, pass a pointer to the [D3D11\\_FEATURE\\_DATA\\_D3D10\\_X\\_HARDWARE\\_OPTIONS](#) structure to the `pFeatureSupportData` parameter, and pass the size of the `D3D11_FEATURE_DATA_D3D10_X_HARDWARE_OPTIONS` structure to the `FeatureSupportDataSize` parameter. `CheckFeatureSupport` returns TRUE in the `ComputeShaders_Plus_RawAndStructuredBuffers_Via_Shader_4_x` member of `D3D11_FEATURE_DATA_D3D10_X_HARDWARE_OPTIONS` if the Direct3D 10.x hardware supports compute shaders.

The [10Level9 Reference](#) section lists the differences between how various [ID3D11Device](#) and [ID3D11DeviceContext](#) methods behave at various 10Level9 feature levels.

- [Unordered Access Views \(UAVs\)](#)
- [Shader Resource Views \(SRVs\)](#)
- [Thread Groups](#)
  - [Thread Group Dimensions](#)
  - [Two-Dimensional Thread Indices](#)
  - [Thread Group Shared Memory \(TGSM\)](#)
- [D3DCompile with D3DCOMPILE\\_SKIP\\_OPTIMIZATION](#)
- [Related topics](#)

## Unordered Access Views (UAVs)

Raw ([RWByteAddressBuffer](#)) and Structured ([RWStructuredBuffer](#)) Unordered Access Views are supported on downlevel hardware, with the following limitations:

- Only a single UAV may be bound to a pipeline at a time through [ID3D11DeviceContext::CSSetUnorderedAccessViews](#).
- The base offset for a Raw UAV must be aligned on a 256-byte boundary (instead of 16-byte alignment required for Direct3D 11 hardware).

Typed UAVs are not supported on downlevel hardware. This includes [Texture1D](#), [Texture2D](#), and [Texture3D](#) UAVs.

Pixel Shaders on downlevel hardware do not support unordered access.

## Shader Resource Views (SRVs)

Raw and Structured Buffers as Shader Resource Views are supported on downlevel hardware for read-only access, as they are on Direct3D 11 hardware. These resource types are supported for Vertex Shaders, Geometry Shaders, Pixel Shaders as well as Compute Shaders.

## Thread Groups

A compute shader can execute on many threads in parallel, within a thread group.

Thread groups are supported on downlevel hardware, with the following limitations:

### Thread Group Dimensions

Thread groups defined for downlevel hardware are limited to X and Y dimensions of 768. This is less than the maximum values of 1024 for Direct3D 11 hardware. The maximum Z dimension of 64 is unchanged.

The total number of threads in the group ( $X \times Y \times Z$ ) is limited to 768. This is less than the limit of 1024 for Direct3D 11 hardware.

If these numbers are exceeded, shader compilation will fail.

### Two-Dimensional Thread Indices

A particular thread within a thread group is indexed using a 3D vector given by (x,y,z).

For compute shaders operating on downlevel hardware, thread groups only support two dimensions. This means that the Z value in the 3D vector must always be 1.

This limitation specifically applies to the following:

- [ID3D11DeviceContext::Dispatch](#)— The *ThreadGroupCountZ* argument must be 1.
- [ID3D11DeviceContext::DispatchIndirect](#)— This function is not supported on downlevel hardware.
- [numthreads](#)— The Z value must be 1.

## Thread Group Shared Memory (TGSM)

Thread Group Shared Memory is limited to 16Kb on downlevel hardware. This is less than the 32Kb that is available to Direct3D 11 hardware.

A Compute Shader thread may only write to its own region of TGSM. This write-only region has a maximum size of 256 bytes or less, with the maximum decreasing as the number of threads declared for the group increases.

The following table defines the per-thread maximum size of a TGSM region for the number of threads in the group:

<b>Number of Threads in Group</b>	<b>Maximum TGSM Size Per Thread</b>
0-64	256
65-68	240
69-72	224
73-76	208
77-84	192
85-92	176
93-100	160
101-112	144
113-128	128
129-144	112
145-168	96
169-204	80
205-256	64
257-340	48
341-512	32

Number of Threads in Group	Maximum TGSM Size Per Thread
513-768	16

A Compute Shader thread may read the TGSM from any location.

## D3DCompile with D3DCOMPILE\_SKIP\_OPTIMIZATION

[D3DCompile](#) returns `E_NOTIMPL` when you pass `cs_4_0` as the shader target along with the [D3DCOMPILE\\_SKIP\\_OPTIMIZATION](#) compile option. The `cs_5_0` shader target works with `D3DCOMPILE_SKIP_OPTIMIZATION`.

## Related topics

[Direct3D 11 on Downlevel Hardware](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Preventing Unwanted NULL Pixel Shader SRVs

Article • 08/23/2019

Direct3D 11 applications that run on Direct3D 9 graphics hardware could inadvertently cause the driver to receive **NULL** shader-resource views (SRVs) even when the applications bind non-**NULL** SRVs to the pixel shader stage. This situation can occur only if the applications destroy SRVs while they execute. This topic discusses how to work around the driver receiving **NULL** shader-resource views (SRVs) even when non-**NULL** SRVs are bound to the pixel shader stage.

To prevent the driver from receiving unwanted **NULL** SRVs, the applications must call [ID3D11DeviceContext::PSSetShaderResources](#) to unset all SRVs before each call to [ID3D11DeviceContext::PSSetShader](#). However, if the applications do not destroy SRVs until the end of their code execution, they do not need to unset the SRVs.

The [10Level9 Reference](#) section lists the differences between how various [ID3D11Device](#) and [ID3D11DeviceContext](#) methods behave at various 10Level9 feature levels.

## Related topics

[Direct3D 11 on Downlevel Hardware](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Using Direct3D 11 feature data to supplement Direct3D feature levels

Article • 08/19/2020

Find out how to check device support for optional features, including features that were added in recent versions of Windows.

[Direct3D feature levels](#) indicate well-defined sets of GPU functionality that roughly correspond to different generations of graphics hardware. This greatly simplifies the task of checking hardware capabilities, and also provides a consistent experience across a wide array of different devices.

To account for some of the variance across different hardware implementations - including legacy hardware, mobile hardware, and modern hardware - some features are considered optional. Support for these features can be determined by calling [ID3D11Device::CheckFeatureSupport](#) and supplying the relevant D3D11\_FEATURE\_DATA\_\* structure. This topic describes the various optional Direct3D 11 features, how some of them work together, and how you can avoid checking for every single optional feature.

## How to check for optional feature support

Call [ID3D11Device::CheckFeatureSupport](#), providing the structure that represents the optional feature you'd like to use. If the method returns **S\_OK**, that means you're on a version of the Direct3D runtime that supports the optional feature. If it returns **E\_INVALIDARG**, that means you're on a version of the Direct3D 11 runtime from before the optional feature was added - this means the optional feature is not available, along with any other optional features introduced in the same version of Direct3D 11 or later.

## Can I minimize the work required for feature support checks?

In addition to having the right Direct3D 11 runtime (usually associated with a Windows version) the graphics driver must also be recent enough to support the optional feature. The WDDM specifications require optional features to be supported if the hardware can support it. So when a graphics driver supports one of the optional features that were added in a particular version of Windows, it usually means that the graphics driver supports the other features added in that version of Windows. For example, if a device

driver supports shadows on feature level 9, then you know the device driver is at least WDDM 1.2.

**Note** If a Microsoft Direct3D device supports [feature level 11.1](#), all of the optional features indicated by [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) are automatically supported except [SAD4ShaderInstructions](#) and [ExtendedDoublesShaderInstructions](#).

The runtime always sets the following groupings of members identically. That is, all the values in a grouping are **TRUE** or **FALSE** together:

- [DiscardAPIsSeenByDriver](#) and [FlagsForUpdateAndCopySeenByDriver](#)
- [ClearView](#), [CopyWithOverlap](#), [ConstantBufferPartialUpdate](#),  
[ConstantBufferOffsetting](#), and [MapNoOverwriteOnDynamicConstantBuffer](#)
- [MapNoOverwriteOnDynamicBufferSRV](#) and  
[MultisampleRTVWithForcedSampleCountOne](#)

**Feature level 11.2 options ([D3D11\\_FEATURE\\_D3D11\\_OPTIONS1](#))**: The optional features indicated by this field are independent and must be checked individually.

## Feature support on Windows RT 8.1 and Windows Phone 8.1 devices

Windows RT tablet devices can support a variety of feature levels and optional features, are optimized for reduced power consumption, and use integrated graphics instead of discrete GPUs. Windows Store apps for ARM devices must support feature level 9.1. DirectX apps for Windows RT should take advantage of optional features that can save power and cycles - such as simple instancing - when they are available.

Windows Phone 8 mobile devices support feature level 9.3 with specific optional features. See [Direct3D feature level 9\\_3 for Windows Phone 8](#).

## What are the Direct3D 11 optional features?

The rest of this article describes the optional features available in Direct3D 11.2. Features are described in chronological order by when they were added so you can get a sense for what features are in different versions of Direct3D 11.

## Optional compute shader support for feature level 10

The following feature is always available for feature level 10 devices:

**D3D11\_FEATURE\_DATA\_D3D10\_X\_HARDWARE\_OPTIONS**: If this is TRUE, the device supports compute shaders. This includes support for raw and structured buffers.

When the feature level 10\_0 or 10\_1 device supports this feature, the device is not guaranteed to support compute shader 4.1. Apps should be prepared to fall back to a compute shader 4.0 if **ID3D11Device::CreateComputeShader** throws an exception with a compute shader 4.1 program.

## Optional capabilities for feature level 9

The following features are added for feature level 9 starting in Windows 8:

**D3D11\_FEATURE\_DATA\_D3D9\_OPTIONS**: Indicates support for wrap texture addressing with non-power-of-2 textures. If this is supported, D3D11\_TEXTURE\_ADDRESS\_MODE\_WRAP can be used with such textures.

**D3D11\_FEATURE\_DATA\_D3D9\_SHADOW\_SUPPORT**: Indicates support for comparison samplers in shader model 4.0 feature level 9\_x shaders. This is used for depth testing in pixel shaders, enabling support for common techniques such as shadow mapping and stencils.

The following feature was added for feature level 9 devices starting in Windows 8.1:

**D3D11\_FEATURE\_DATA\_D3D9\_SIMPLE\_INSTANCING\_SUPPORT**: Indicates support for simple instancing features that might be available on DirectX 9-level hardware. Simple instancing means that all **InstanceDataStepRate** members of the **D3D11\_INPUT\_ELEMENT\_DESC** structures used to define the input layout must be equal to 1. Devices that support feature level 9.3 or higher already include full support for instancing.

## Optional floating-point precision support for shader programs

**D3D11\_FEATURE\_DATA\_SHADER\_MIN\_PRECISION\_SUPPORT**: The fields in this struct indicate the length of floating-point numbers when minimum precision is enabled, or 0 if only full 32-bit floating point precision is supported.

For feature level 9 devices, the minimum precision for the vertex shader can be different from the pixel shader. The precision for the vertex shader is indicated in the **AllOtherShaderStagesMinPrecision** field.

**D3D11\_FEATURE\_DATA\_DOUBLEs**: Feature level 11 devices can support double precision calculations within shader model 5.0 programs. Support for double precision calculations within the shader means that floats can be converted to doubles within the compute shader program, providing the benefit of higher precision computation within each shader pass. The double-precision numbers must be converted back to floats before being written to the output buffer. Note that double-precision division is not necessarily supported.

## Additional capabilities for Direct3D 11.2

Direct3D 11.2 adds four new optional features that can be supported by Direct3D 11 devices. These features are in the **D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS1** structure:

**TiledResourcesTier**: Indicates support for tiled resources, and indicates the tier level supported.

**MinMaxFiltering**: Indicates support for D3D11\_FILTER\_MINIMUM\_\* and D3D11\_FILTER\_MAXIMUM\_\* filtering options, which compare the filtering result to the minimum (or maximum) value. See [D3D11\\_FILTER](#).

**ClearViewAlsoSupportsDepthOnlyFormats**: Indicates support for clearing depth buffer resource views.

**MapOnDefaultBuffers**: Indicates support for mapping render target buffers created with the **D3D11\_USAGE\_DEFAULT** flag.

## Tile based rendering

**D3D11\_FEATURE\_DATA\_ARCHITECTURE\_INFO**: Indicates whether the graphics device batches rendering commands, and performs tile-based rendering by default. This can be used as a hint for graphics engine optimization.

## Optional features for development and debugging

**D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS::DiscardAPIsSeenByDriver**: You can monitor this member during development to rule out legacy drivers on hardware where [DiscardView](#) and [DiscardResource](#) might have otherwise been beneficial.

**D3D11\_FEATURE\_DATA\_MARKER\_SUPPORT**: This is supported if the hardware and driver support data marking for GPU profiling.

# Related topics

[Devices](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Resources

Article • 07/06/2022

Resources provide data to the pipeline and define what is rendered during your scene. Resources can be loaded from your game media or created dynamically at run time. Typically, resources include texture data, vertex data, and shader data. Most Direct3D applications create and destroy resources extensively throughout their lifespan. This section describes aspects of Direct3D 11 resources.

## In this section

Topic	Description
<a href="#">Introduction to a Resource in Direct3D 11</a>	This topic introduces Direct3D resources such as buffers and textures.
<a href="#">Types of Resources</a>	This topic describes the types of resources from Direct3D 10, as well as new types in Direct3D 11 including structured buffers and writable textures and buffers.
<a href="#">Resource Limits</a>	This topic contains a list of resources that Direct3D 11 supports (specifically <a href="#">feature level</a> 11 or 9.x hardware).
<a href="#">Subresources</a>	This topic describes texture subresources, or portions of a resource.
<a href="#">Buffers</a>	Buffers contain data that is used for describing geometry, indexing geometry information, and shader constants. This section describes buffers that are used in Direct3D 11 and links to task-based documentation for common scenarios.
<a href="#">Textures</a>	This section describes textures that are used in Direct3D 11 and links to task-based documentation for common scenarios.
<a href="#">Floating-point rules</a>	Direct3D 11 supports several floating-point representations. All floating-point computations operate under a defined subset of the IEEE 754 32-bit single precision floating-point rules.
<a href="#">Tiled resources</a>	Tiled resources can be thought of as large logical resources that use small amounts of physical memory.
<a href="#">Displayable surfaces</a>	The displayable surfaces feature means that buffers that are presented may have varying properties, and you may present them in any order.

## Related topics

## Feedback

Was this page helpful?

 Yes

 No

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

# Introduction to a Resource in Direct3D

## 11

Article • 10/06/2021

Resources are the building blocks of your scene. They contain most of the data that Direct3D uses to interpret and render your scene. Resources are areas in memory that can be accessed by the Direct3D pipeline. Resources contain the following types of data: geometry, textures, shader data. This topic introduces Direct3D resources such as buffers and textures.

You can create resources that are strongly typed or typeless; you can control whether resources have both read and write access; you can make resources accessible to only the CPU, GPU, or both. Up to 128 resources can be active for each pipeline stage.

Direct3D guarantees to return zero for any resource that is accessed out of bounds.

The lifecycle of a Direct3D resource is:

- Create a resource using one of the create methods of the [ID3D11Device](#) interface.
- Bind a resource to the pipeline using a context and one of the set methods of the [ID3D11DeviceContext](#) interface.
- Deallocate a resource by calling the [Release](#) method of the resource interface.

This section contains the following topics:

- [Strong vs Weak Typing](#)
- [Resource Views](#)
- [Raw Views of Buffers](#)
- [Related topics](#)

## Strong vs Weak Typing

There are two ways to fully specify the layout (or memory footprint) of a resource:

- Typed - fully specify the type when the resource is created.
- Typeless - fully specify the type when the resource is bound to the pipeline.

Creating a fully-typed resource restricts the resource to the format it was created with. This enables the runtime to optimize access, especially if the resource is created with flags indicating that it cannot be mapped by the application. Resources created with a specific type cannot be reinterpreted using the view mechanism unless the resource was

created with the D3D10\_DDI\_BIND\_PRESENT flag. If D3D10\_DDI\_BIND\_PRESENT is set, then render-target or shader resource views can be created on these resources using any of the fully typed members of the appropriate family, even if the original resource was created as fully typed.

In a type less resource, the data type is unknown when the resource is first created. The application must choose from the available type less formats (see [DXGI\\_FORMAT](#)). You must specify the size of the memory to allocate and whether the runtime will need to generate the subtextures in a mipmap. However, the exact data format (whether the memory will be interpreted as integers, floating point values, unsigned integers etc.) is not determined until the resource is bound to the pipeline with a [resource view](#). As the texture format remains flexible until the texture is bound to the pipeline, the resource is referred to as weakly typed storage. Weakly typed storage has the advantage that it can be reused or reinterpreted in another format as long as the number of components and the bit count of each component are the same in both formats.

A single resource can be bound to multiple pipeline stages as long as each has a unique view, which fully qualifies the formats at each location. For example, a resource created with the format DXGI\_FORMAT\_R32G32B32A32\_TYPELESS could be used as a DXGI\_FORMAT\_R32G32B32A32\_FLOAT and a DXGI\_FORMAT\_R32G32B32A32\_UINT at different locations in the pipeline simultaneously.

## Resource Views

Resources can be stored in general purpose memory formats so that they can be shared by multiple pipeline stages. A pipeline stage interprets resource data using a view. A resource view is conceptually similar to casting the resource data so that it can be used in a particular context.

A view can be used with a typeless resource. That is, you can create a resource at compile time and declare the data type when the resource is bound to the pipeline. A view created for a typeless resource always has the same number of bits per component; the way the data is interpreted is dependent on the format specified. The format specified must be from the same family as the typeless format used when creating the resource. For example, a resource created with the R8G8B8A8\_TYPELESS format cannot be viewed as a R32\_FLOAT resource even though both formats may be the same size in memory.

A view also exposes other capabilities such as the ability to read back depth/stencil surfaces in a shader, generating a dynamic cubemap in a single pass, and rendering simultaneously to multiple slices of a volume.

Resource Interface	Description
<a href="#">ID3D11DepthStencilView</a>	Access a texture resource during depth-stencil testing.
<a href="#">ID3D11RenderTargetView</a>	Access a texture resource that is used as a render-target.
<a href="#">ID3D11ShaderResourceView</a>	Access a shader resource such as a constant buffer, a texture buffer, a texture or a sampler.
<a href="#">ID3D11UnorderedAccessView</a>	Access an unordered resource using a pixel shader or a compute shader.

## Raw Views of Buffers

You can think of a raw buffer, which can also be called a [byte address buffer](#), as a bag of bits to which you want raw access, that is, a buffer that you can conveniently access through chunks of one to four 32-bit typeless address values. You indicate that you want raw access to a buffer (or, a raw view of a buffer) when you call one of the following methods to create a view to the buffer:

- To create a shader resource view (SRV) to the buffer, call [ID3D11Device::CreateShaderResourceView](#) with the flag [D3D11\\_BUFFEREX\\_SRV\\_FLAG\\_RAW](#). You specify this flag in the **Flags** member of the [D3D11\\_BUFFEREX\\_SRV](#) structure. You set [D3D11\\_BUFFEREX\\_SRV](#) in the **BufferEx** member of the [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure to which the *pDesc* parameter of [ID3D11Device::CreateShaderResourceView](#) points. You also set the [D3D11\\_SRV\\_DIMENSION\\_BUFFEREX](#) value in the **ViewDimension** member of [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) to indicate that the SRV is a raw view.
- To create an unordered access view (UAV) to the buffer, call [ID3D11Device::CreateUnorderedAccessView](#) with the flag [D3D11\\_BUFFER\\_UAV\\_FLAG\\_RAW](#). You specify this flag in the **Flags** member of the [D3D11\\_BUFFER\\_UAV](#) structure. You set [D3D11\\_BUFFER\\_UAV](#) in the **Buffer** member of the [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure to which the *pDesc* parameter of [ID3D11Device::CreateUnorderedAccessView](#) points. You also set the [D3D11\\_UAV\\_DIMENSION\\_BUFFER](#) value in the **ViewDimension** member of [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) to indicate that the UAV is a raw view.

You can use the HLSL [ByteAddressBuffer](#) and [RWByteAddressBuffer](#) object types when you work with raw buffers.

To create a raw view to a buffer, you must first call [ID3D11Device::CreateBuffer](#) with the [D3D11\\_RESOURCE\\_MISC\\_BUFFER\\_ALLOW\\_RAW\\_VIEWS](#) flag to create the underlying buffer resource. You specify this flag in the `MiscFlags` member of the [D3D11\\_BUFFER\\_DESC](#) structure to which the `pDesc` parameter of `ID3D11Device::CreateBuffer` points. You can't combine the [D3D11\\_RESOURCE\\_MISC\\_BUFFER\\_ALLOW\\_RAW\\_VIEWS](#) flag with [D3D11\\_RESOURCE\\_MISC\\_BUFFER\\_STRUCTURED](#). Also, if you specify [D3D11\\_BIND\\_CONSTANT\\_BUFFER](#) in `BindFlags` of [D3D11\\_BUFFER\\_DESC](#), you can't also specify [D3D11\\_RESOURCE\\_MISC\\_BUFFER\\_ALLOW\\_RAW\\_VIEWS](#) in `MiscFlags`. This is not a limitation of just raw views because constant buffers already have a constraint that they can't be combined with any other view.

Other than the preceding invalid cases, when you create a buffer with [D3D11\\_RESOURCE\\_MISC\\_BUFFER\\_ALLOW\\_RAW\\_VIEWS](#), you aren't limited in functionality versus not setting [D3D11\\_RESOURCE\\_MISC\\_BUFFER\\_ALLOW\\_RAW\\_VIEWS](#). That is, you can use such a buffer for non-raw access in any number of ways that are possible with Direct3D. If you specify the [D3D11\\_RESOURCE\\_MISC\\_BUFFER\\_ALLOW\\_RAW\\_VIEWS](#) flag, you only increase the available functionality.

## Related topics

[Resources](#)

[New Resource Types](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Types of Resources

Article • 04/26/2022

Resources are areas in memory that can be accessed by the Direct3D pipeline, they are the building blocks of your scene. Resources contain many types of data such as geometry, textures or shader data that Direct3D uses to populate and render your scene. This topic describes the types of resources from Direct3D 10, as well as new types in Direct3D 11 including structured buffers and writable textures and buffers.

## Note: Differences between Direct3D 11 and Direct3D 10

Direct3D 11 supports several new resource types including:

- [read-write buffers and textures](#)
- [structured buffers](#)
- [byte-address buffers](#)
- [append and consume buffers](#)
- [unordered access buffer or texture](#)

Direct3D 11.2 supports [tiled resources](#).

Both Direct3D 10 and Direct3D 11 support the [buffer](#) and [texture](#) types that were introduced in Direct3D 10.

## Related topics

[Resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Resource Limits (Direct3D 11)

Article • 08/19/2020

This topic contains a list of resources that Direct3D 11 supports (specifically [feature level 11](#) or 9.x hardware).

- [Resource limits for feature level 11 hardware](#)
- [Resource limits for feature level 9.x hardware](#)
- [Related topics](#)

## Resource limits for feature level 11 hardware

All of these resource limits are defined as constants in D3d11.h.

Resource	Limit
Number of elements in a constant buffer	D3D11_REQ_CONSTANT_BUFFER_ELEMENT_COUNT (4096)
Number of texels (independent of struct size) in a buffer	D3D11_REQ_BUFFER_RESOURCE_TEXEL_COUNT_2_TO_EXP (2 <sup>7</sup> ) texels
Texture1D U dimension	D3D11_REQ_TEXTURE1D_U_DIMENSION (16384)
Texture1DArray dimension	D3D11_REQ_TEXTURE1D_ARRAY_AXIS_DIMENSION (2048 array slices)
Texture2D U/V dimension	D3D11_REQ_TEXTURE2D_U_OR_V_DIMENSION (16384)
Texture2DArray dimension	D3D11_REQ_TEXTURE2D_ARRAY_AXIS_DIMENSION (2048 array slices)
Texture3D U/V/W dimension	D3D11_REQ_TEXTURE3D_U_V_OR_W_DIMENSION (2048)
TextureCube dimension	D3D11_REQ_TEXTURECUBE_DIMENSION (16384)
Resource size (in MB) for any of the preceding resources	min(max(128,0.25f * (amount of dedicated VRAM)), 2048) MB D3D11_REQ_RESOURCE_SIZE_IN_MEGABYTES_EXPRESSION_A_TERM (128) D3D11_REQ_RESOURCE_SIZE_IN_MEGABYTES_EXPRESSION_B_TERM (0.25f) D3D11_REQ_RESOURCE_SIZE_IN_MEGABYTES_EXPRESSION_C_TERM (2048)

Resource	Limit
Anisotropic filtering maxanisotropy	D3D11_REQ_MAXANISOTROPY (16)
Resource dimension addressable by filtering hardware	D3D11_REQ_FILTERING_HW_ADDRESSABLE_RESOURCE_DIMENSION (16384) per dimension
Resource size (in MB) addressable by IA (input or vertex data) or VS/GS/PS (point sample)	max(128,0.25f * (amount of dedicated VRAM)) MB D3D11_REQ_RESOURCE_SIZE_IN_MEGABYTES_EXPRESSION_A_TERM (128) D3D11_REQ_RESOURCE_SIZE_IN_MEGABYTES_EXPRESSION_B_TERM (0.25f)
Total number of resource views per context (Each array counts as 1) (all view types have shared limit)	D3D11_REQ_RESOURCE_VIEW_COUNT_PER_DEVICE_2_TO_EXP (2 )
Buffer structure size (multi-element)	D3D11_REQ_MULTI_ELEMENT_STRUCTURE_SIZE_IN_BYTES (2048 bytes)
Stream output size	Same as the number of texels in a buffer (see preceding)
Draw or DrawInstanced vertex count (including instancing)	D3D11_REQ_DRAW_VERTEX_COUNT_2_TO_EXP (2 )
DrawIndexed[Instanced] () vertex count (including instancing)	D3D11_REQ_DRAWINDEXED_INDEX_COUNT_2_TO_EXP (2 )
GS invocation output data (components * vertices)	D3D11_GS_MAX_OUTPUT_VERTEX_COUNT_ACROSS_INSTANCES (1024)
Total number of sampler objects per context	D3D11_REQ_SAMPLER_OBJECT_COUNT_PER_DEVICE (4096)
Total number of viewport/scissor objects per pipeline	D3D11_VIEWPORT_AND_SCISSORRECT_OBJECT_COUNT_PER_PIPELINE (16)
Total number of clip/cull distances per vertex	D3D11_CLIP_OR_CULL_DISTANCE_COUNT (8)

<b>Resource</b>	<b>Limit</b>
Total number of blend objects per context	D3D11_REQ_BLEND_OBJECT_COUNT_PER_DEVICE (4096)
Total number of depth/stencil objects per context	D3D11_REQ_DEPTH_STENCIL_OBJECT_COUNT_PER_DEVICE (4096)
Total number of rasterizer state objects per context	D3D11_REQ_RASTERIZER_OBJECT_COUNT_PER_DEVICE (4096)
Maximum per-pixel sample count during multisampling	D3D11_MAX_MULTISAMPLE_SAMPLE_COUNT (32)
Shader resource vertex-element count (four 32-bit components)	D3D11_STANDARD_VERTEX_ELEMENT_COUNT (32)
Common-shader core (four 32-bit components) temp-register count ( $r\# + \text{indexable } x\#[n]$ )	D3D11_COMMONSHADER_TEMP_REGISTER_COUNT (4096)
Common-shader core constant-buffer slots	D3D11_COMMONSHADER_CONSTANT_BUFFER_HW_SLOT_COUNT (15) (+1 set aside for an immediate constant buffer in shaders)
Common-shader core input-resource slots	D3D11_COMMONSHADER_INPUT_RESOURCE_REGISTER_COUNT (128)
Common-shader core sampler slots	D3D11_COMMONSHADER_SAMPLER_SLOT_COUNT (16)
Common-shader core subroutine-nesting limit	D3D11_COMMONSHADER_SUBROUTINE_NESTING_LIMIT (32)
Common-shader core flow-control nesting limit	D3D11_COMMONSHADER_FLOWCONTROL_NESTING_LIMIT (64)
Vertex shader input-register count (four 32-bit components)	D3D11_VS_INPUT_REGISTER_COUNT (32)
Vertex shader output-register count (four 32-bit components)	D3D11_VS_OUTPUT_REGISTER_COUNT (32)

<b>Resource</b>	<b>Limit</b>
Geometry shader input-register count (four 32-bit components)	D3D11_GS_INPUT_REGISTER_COUNT (32)
Geometry shader output-register count (four 32-bit components)	D3D11_GS_OUTPUT_REGISTER_COUNT (32)
Pixel shader input-register count (four 32-bit components)	D3D11_PS_INPUT_REGISTER_COUNT (32)
Pixel shader output slots	8
Pixel shader output depth register count (one 32-bit component)	D3D11_PS_OUTPUT_DEPTH_REGISTER_COUNT (1)
Input assembler index input resource slots	D3D11_IA_INDEX_INPUT_RESOURCE_SLOT_COUNT (1)
Input assembler vertex input resource slots	D3D11_IA_VERTEX_INPUT_RESOURCE_SLOT_COUNT (32)
Hull shader control point input-register count (four 32-bit components)	D3D11_HS_CONTROL_POINT_PHASE_INPUT_REGISTER_COUNT (32)
Hull shader number of input control points	D3D11_HS_CONTROL_POINT_REGISTER_COMPONENT_BIT_COUNT (32)
Hull shader control point output-register count (four 32-bit components)	D3D11_HS_CONTROL_POINT_PHASE_OUTPUT_REGISTER_COUNT (32)
Hull shader number of output control points	D3D11_HS_CONTROL_POINT_REGISTER_COMPONENT_BIT_COUNT (32)
Hull shader patch constant output-register count (four 32-bit components)	D3D11_HS_OUTPUT_PATCH_CONSTANT_REGISTER_COUNT (32)

Resource	Limit
Domain shader control point input-register count (four 32-bit components)	D3D11_DS_INPUT_CONTROL_POINT_REGISTER_COUNT (32)
Domain shader number of input control points	D3D11_DS_INPUT_CONTROL_POINT_COMPONENT_BIT_COUNT (32)
Domain shader patch constant input-register count (four 32-bit components)	D3D11_DS_INPUT_PATCH_CONSTANT_REGISTER_COUNT (32)
Domain shader tessellated vertex output-register count (four 32-bit components)	D3D11_DS_OUTPUT_REGISTER_COUNT (32)
Compute shader unordered access view (UAV) slots	D3D11_PS_CS_UAV_REGISTER_COUNT (8)4
Resource tile size in bytes	D3D11_2_TILED_RESOURCE_TILE_SIZE_IN_BYTES ( 65536 )

An app can try to allocate more memory for a resource than the maximum resource size specifies. That is, the Direct3D 11 runtime allows these memory allocation attempts in the event that the hardware might support them. However, the Direct3D 11 runtime only guarantees that allocations within the maximum resource size are supported by all [feature level](#) 11 hardware. If an app tries to allocate memory for a resource within the maximum resource size, the runtime fails the attempt only if the operating system runs out of resources. If an app tries to allocate memory for a resource above the maximum resource size, the runtime can fail the attempt because either the operating system is overextended or the hardware does not support allocations above the maximum resource size. The debug layer only checks

D3D11\_REQ\_RESOURCE\_SIZE\_IN\_MEGABYTES\_EXPRESSION\_A\_TERM (128) MB.

The pixel shader output slots are shared between pixel output registers (four 32-bit components) and UAVs.

The total number of components for all hull shader to domain shader control points is limited to 3968, which is 128 less than the maximum control points times the maximum control point registers times four components.

For compute shader profiles CS\_4\_0 and CS\_4\_1 there is only 1 UAV available. For more information about shader profiles, see [Shader Model 5](#).

## Resource limits for feature level 9.x hardware

All of these 9.x [feature level](#) resource limits are defined as constants in D3dcommon.h.

Resource	Limit
Feature level 9_1 texture1D U dimension	D3D_FL9_1_REQ_TEXTURE1D_U_DIMENSION (2048)
Feature level 9_3 texture1D U dimension	D3D_FL9_3_REQ_TEXTURE1D_U_DIMENSION (4096)
Feature level 9_1 texture2D U/V dimension	D3D_FL9_1_REQ_TEXTURE2D_U_OR_V_DIMENSION (2048)
Feature level 9_3 texture2D U/V dimension	D3D_FL9_3_REQ_TEXTURE2D_U_OR_V_DIMENSION (4096)
Feature level 9_1 texture3D U/V/W dimension	D3D_FL9_1_REQ_TEXTURE3D_U_V_OR_W_DIMENSION (256)
Feature level 9_1 textureCube dimension	D3D_FL9_1_REQ_TEXTURECUBE_DIMENSION (512)
Feature level 9_3 textureCube dimension	D3D_FL9_3_REQ_TEXTURECUBE_DIMENSION (4096)
Feature level 9_1 anisotropic filtering maxanisotropy default	D3D_FL9_1_DEFAULT_MAX_ANISOTROPY (2)
Feature level 9_1 maximum input assembler primitives	D3D_FL9_1_IA_PRIMITIVE_MAX_COUNT (65535)
Feature level 9_2 maximum input assembler primitives	D3D_FL9_2_IA_PRIMITIVE_MAX_COUNT (1048575)
Feature level 9_1 simultaneous render targets	D3D_FL9_1_SIMULTANEOUS_RENDER_TARGET_COUNT (1)
Feature level 9_3 simultaneous render targets	D3D_FL9_3_SIMULTANEOUS_RENDER_TARGET_COUNT (4)
Feature level 9_1 maximum texture repeat	D3D_FL9_1_MAX_TEXTURE_REPEAT (128)
Feature level 9_2 maximum texture repeat	D3D_FL9_2_MAX_TEXTURE_REPEAT (2048)
Feature level 9_3 maximum texture repeat	D3D_FL9_3_MAX_TEXTURE_REPEAT (8192)

# Related topics

Resources

---

## Feedback

Was this page helpful?

 Yes

 No

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

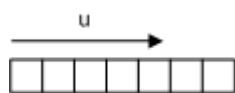
# Subresources (Direct3D 11 Graphics)

Article • 12/10/2020

This topic describes texture subresources, or portions of a resource.

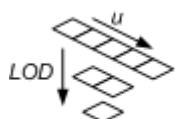
Direct3D can reference an entire resource or it can reference subsets of a resource. The term subresource refers to a subset of a resource.

A buffer is defined as a single subresource. Textures are a little more complicated because there are several different texture types (1D, 2D, etc.) some of which support mipmap levels and/or texture arrays. Beginning with the simplest case, a 1D texture is defined as a single subresource, as shown in the following illustration.

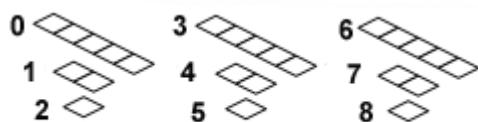


This means that the array of texels that make up a 1D texture are contained in a single subresource.

If you expand a 1D texture with three mipmap levels, it can be visualized like the following illustration.



Think of this as a single texture that is made up of three subresources. A subresource can be indexed using the level-of-detail (LOD) for a single texture. When using an array of textures, accessing a particular subresource requires both the LOD and the particular texture. Alternately, the API combines these two pieces of information into a single zero-based subresource index, as shown in the following illustration.



## Selecting Subresources

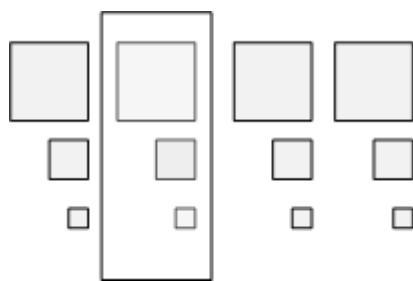
Some APIs access an entire resource (for example the [ID3D11DeviceContext::CopyResource](#) method), others access a portion of a resource

(for example the [ID3D11DeviceContext::UpdateSubresource](#) method or the [ID3D11DeviceContext::CopySubresourceRegion](#) method). The methods that access a portion of a resource generally use a view description (such as the [D3D11\\_TEX2D\\_ARRAY\\_DSV](#) structure) to specify the subresources to access.

The illustrations in the following sections show the terms used by a view description when accessing an array of textures.

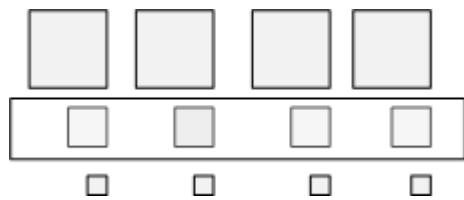
## Array Slice

Given an array of textures, each texture with mipmaps, an *array slice* (represented by the white rectangle) includes one texture and all of its subresources, as shown in the following illustration.



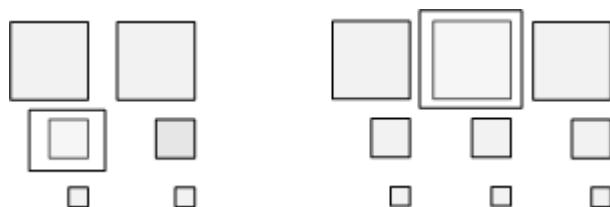
## Mip Slice

A *mip slice* (represented by the white rectangle) includes one mipmap level for every texture in an array, as shown in the following illustration.



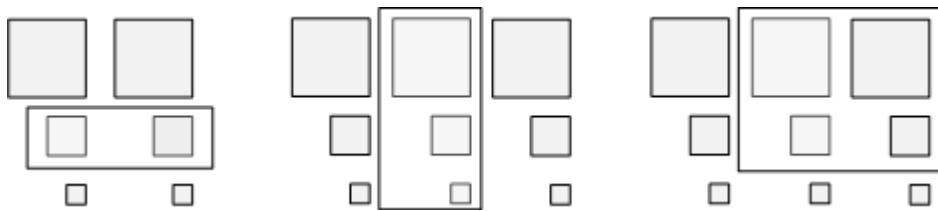
## Selecting a Single Subresource

You can use these two types of slices to choose a single subresource, as shown in the following illustration.



## Selecting Multiple Subresources

Or you can use these two types of slices with the number of mipmap levels and/or number of textures, to choose multiple subresources, as shown in the following illustration.



### ① Note

A **render-target view** can only use a single subresource or mip slice and cannot include subresources from more than one mip slice. That is, every texture in a render-target view must be the same size. A **shader-resource view** can select any rectangular region of subresources, as shown in the figure.

## Related topics

[Resources](#)

## Feedback

Was this page helpful?

Yes

No

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

# Buffers

Article • 08/23/2019

Buffers contain data that is used for describing geometry, indexing geometry information, and shader constants. This section describes buffers that are used in Direct3D 11 and links to task-based documentation for common scenarios.

## In this section

Topic	Description
<a href="#">Introduction to Buffers in Direct3D 11</a>	A buffer resource is a collection of fully typed data grouped into elements. You can use buffers to store a wide variety of data, including position vectors, normal vectors, texture coordinates in a vertex buffer, indexes in an index buffer, or device state. A buffer element is made up of 1 to 4 components. Buffer elements can include packed data values (like R8G8B8A8 surface values), single 8-bit integers, or four 32-bit floating point values.

## Related topics

[How to: Create a Constant Buffer](#)

[How to: Create an Index Buffer](#)

[How to: Create a Vertex Buffer](#)

[Resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Introduction to Buffers in Direct3D 11

Article • 10/31/2022

A buffer resource is a collection of fully typed data grouped into elements. You can use buffers to store a wide variety of data, including position vectors, normal vectors, texture coordinates in a vertex buffer, indexes in an index buffer, or device state. A buffer element is made up of 1 to 4 components. Buffer elements can include packed data values (like R8G8B8A8 surface values), single 8-bit integers, or four 32-bit floating point values.

A buffer is created as an unstructured resource. Because it is unstructured, a buffer cannot contain any mipmap levels, it cannot get filtered when read, and it cannot be multisampled.

## Buffer Types

The following are the buffer resource types supported by Direct3D 11. All buffer types are encapsulated by the [ID3D11Buffer](#) interface.

- [Vertex Buffer](#)
- [Index Buffer](#)
- [Constant Buffer](#)

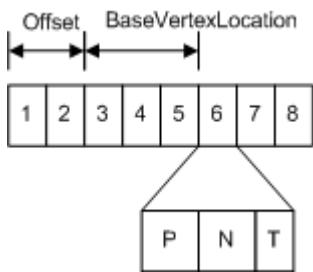
## Vertex Buffer

A vertex buffer contains the vertex data used to define your geometry. Vertex data includes position coordinates, color data, texture coordinate data, normal data, and so on.

The simplest example of a vertex buffer is one that only contains position data. It can be visualized like the following illustration.



More often, a vertex buffer contains all the data needed to fully specify 3D vertices. An example of this could be a vertex buffer that contains per-vertex position, normal and texture coordinates. This data is usually organized as sets of per-vertex elements, as shown in the following illustration.



This vertex buffer contains per-vertex data; each vertex stores three elements (position, normal, and texture coordinates). The position and normal are each typically specified using three 32-bit floats (DXGI\_FORMAT\_R32G32B32\_FLOAT) and the texture coordinates using two 32-bit floats (DXGI\_FORMAT\_R32G32\_FLOAT).

To access data from a vertex buffer you need to know which vertex to access, plus the following additional buffer parameters:

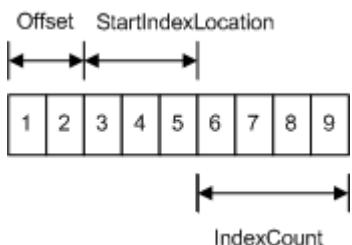
- Offset - the number of bytes from the start of the buffer to the data for the first vertex. You can specify the offset using the [ID3D11DeviceContext::IASetVertexBuffers](#) method.
- BaseVertexLocation - a value added to each index before reading a vertex from the vertex buffer.

Before you create a vertex buffer, you need to define its layout by creating an [ID3D11InputLayout](#) interface; this is done by calling the [ID3D11Device::CreateInputLayout](#) method. After the input-layout object is created, you can bind it to the input-assembler stage by calling the [ID3D11DeviceContext::IASetInputLayout](#).

To create a vertex buffer, call [ID3D11Device::CreateBuffer](#).

## Index Buffer

Index buffers contain integer offsets into vertex buffers and are used to render primitives more efficiently. An index buffer contains a sequential set of 16-bit or 32-bit indices; each index is used to identify a vertex in a vertex buffer. An index buffer can be visualized like the following illustration.



The sequential indices stored in an index buffer are located with the following parameters:

- Offset - the number of bytes from the base address of the index buffer. The offset is supplied to the [ID3D11DeviceContext::IASetIndexBuffer](#) method.
- StartIndexLocation - specifies the first index buffer element from the base address and the offset provided in [IASetIndexBuffer](#). The start location is supplied to the [ID3D11DeviceContext::DrawIndexed](#) or [ID3D11DeviceContext::DrawIndexedInstanced](#) method and represents the first index to render.
- IndexCount - the number of indices to render. The number is supplied to the [DrawIndexed](#) method

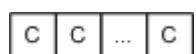
Start of Index Buffer = Index Buffer Base Address + Offset (bytes) + StartIndexLocation \* ElementSize (bytes);

In this calculation, ElementSize is the size of each index buffer element, which is either two or four bytes.

To create an index buffer, call [ID3D11Device::CreateBuffer](#).

## Constant Buffer

A constant buffer allows you to efficiently supply shader constants data to the pipeline. You can use a constant buffer to store the results of the stream-output stage. Conceptually, a constant buffer looks just like a single-element vertex buffer, as shown in the following illustration.



Each element stores a 1-to-4 component constant, determined by the format of the data stored. To create a shader-constant buffer, call [ID3D11Device::CreateBuffer](#) and specify the [D3D11\\_BIND\\_CONSTANT\\_BUFFER](#) member of the [D3D11\\_BIND\\_FLAG](#) enumerated type.

A constant buffer can only use a single bind flag ([D3D11\\_BIND\\_CONSTANT\\_BUFFER](#)), which cannot be combined with any other bind flag. To bind a shader-constant buffer to the pipeline, call one of the following methods:

[ID3D11DeviceContext::GSSetConstantBuffers](#),  
[ID3D11DeviceContext::PSSetConstantBuffers](#), or  
[ID3D11DeviceContext::VSSetConstantBuffers](#).

To read a shader-constant buffer from a shader, use a HLSL load function (for example, [Load](#)). Each shader stage allows up to 15 shader-constant buffers; each buffer can hold up to 4096 constants.

## Related topics

[Buffers](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Textures

Article • 08/23/2019

A texture stores texel information. This section describes textures that are used in Direct3D 11 and links to task-based documentation for common scenarios.

## In this section

Topic	Description
<a href="#">Introduction</a>	A texture resource is a structured collection of data designed to store texels. A texel represents the smallest unit of a texture that can be read or written to by the pipeline. Unlike buffers, textures can be filtered by texture samplers as they are read by shader units. The type of texture impacts how the texture is filtered. Each texel contains 1 to 4 components, arranged in one of the DXGI formats defined by the DXGI_FORMAT enumeration.
<a href="#">Texture Block Compression in Direct3D 11</a>	Block Compression (BC) support for textures has been extended in Direct3D 11 to include the BC6H and BC7 algorithms. BC6H supports high-dynamic range color source data, and BC7 provides better-than-average quality compression with less artifacts for standard RGB source data.

## Related topics

[How to: Create a Texture](#)

[How to: Initialize a Texture Programmatically](#)

[How to: Initialize a Texture From a File](#)

[Resources](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Introduction To Textures in Direct3D 11

Article • 08/23/2019

A texture resource is a structured collection of data designed to store texels. A texel represents the smallest unit of a texture that can be read or written to by the pipeline. Unlike buffers, textures can be filtered by texture samplers as they are read by shader units. The type of texture impacts how the texture is filtered. Each texel contains 1 to 4 components, arranged in one of the DXGI formats defined by the DXGI\_FORMAT enumeration.

Textures are created as a structured resource with a known size. However, each texture may be typed or typeless when the resource is created as long as the type is fully specified using a view when the texture is bound to the pipeline.

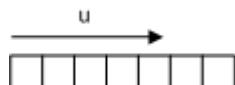
## Texture Types

There are several types of textures: 1D, 2D, 3D, each of which can be created with or without mipmaps. Direct3D 11 also supports texture arrays and multisampled textures.

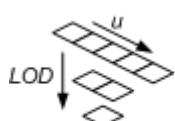
- [1D Textures](#)
- [1D Texture Arrays](#)
- [2D Textures and 2D Texture Arrays](#)
- [3D Textures](#)

## 1D Textures

A 1D texture in its simplest form contains texture data that can be addressed with a single texture coordinate; it can be visualized as an array of texels, as shown in the following illustration. 1D textures are represented by the [ID3D11Texture1D](#) interface.



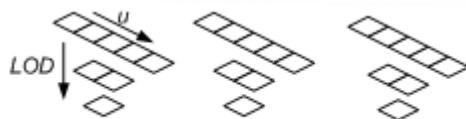
Each texel contains a number of color components depending on the format of the data stored. Adding more complexity, you can create a 1D texture with mipmap levels, as shown in the following illustration.



A mipmap level is a texture that is a power-of-two smaller than the level above it. The topmost level contains the most detail, each subsequent level is smaller. For a 1D mipmap, the smallest level contains one texel. Furthermore, MIP levels always reduce down to 1:1. When mipmaps are generated for an odd sized texture, the next lower level is always even size (except when the lowest level reaches 1). For example, the diagram illustrates a 5x1 texture whose next lowest level is a 2x1 texture, whose next (and last) mip level is a 1x1 sized texture. The levels are identified by an index called a LOD (level-of-detail) which is used to access the smaller texture when rendering geometry that is not as close to the camera.

## 1D Texture Arrays

Direct3D 11 also supports arrays of textures. A 1D texture array is also represented by the [ID3D11Texture1D](#) interface. An array of 1D textures looks conceptually like the following illustration.



This texture array contains three textures. Each of the three textures has a texture width of 5 (which is the number of elements in the first layer). Each texture also contains a 3 layer mipmap.

All texture arrays in Direct3D are a homogeneous array of textures; this means that every texture in a texture array must have the same data format and size (including texture width and number of mipmap levels). You may create texture arrays of different sizes, as long as all the textures in each array match in size.

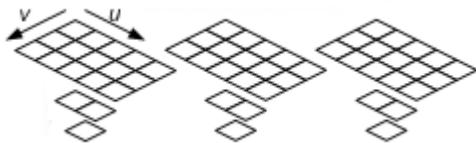
## 2D Textures and 2D Texture Arrays

A Texture2D resource contains a 2D grid of texels. Each texel is addressable by a u, v vector. Since it is a texture resource, it may contain mipmap levels, and subresources. 2D textures are represented by the [ID3D11Texture2D](#) interface. A fully populated 2D texture resource looks like the following illustration.



This texture resource contains a single 3x5 texture with three mipmap levels.

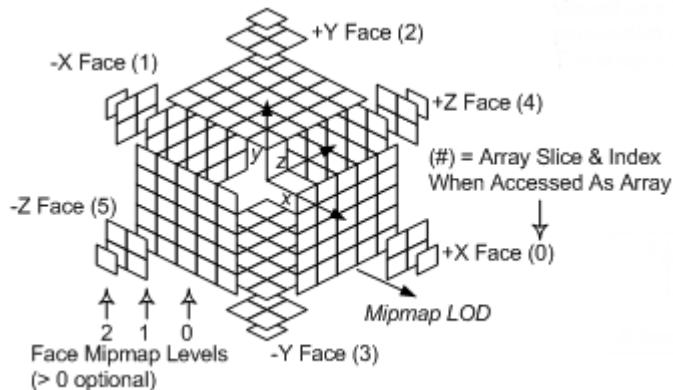
A 2D texture array resource is a homogeneous array of 2D textures; that is, each texture has the same data format and dimensions (including mipmap levels). A 2D texture array is also represented by the [ID3D11Texture2D](#) interface. It has a similar layout as the 1D texture array except that the textures now contain 2D data, as shown in the following illustration.



This texture array contains three textures; each texture is 3x5 with two mipmap levels.

## Using a 2D Texture Array as a Texture Cube

A texture cube is a 2D texture array that contains 6 textures, one for each face of the cube. A fully populated texture cube looks like the following illustration.



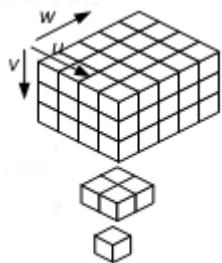
A 2D texture array that contains 6 textures may be read from within shaders with the cube map intrinsic functions, after they are bound to the pipeline with a cube-texture view. Texture cubes are addressed from the shader with a 3D vector pointing out from the center of the texture cube.

### ⓘ Note

Devices that you create with **10\_1 feature level** and above can support arrays of texture cubes in which the number of textures is equal to the number of texture cubes in an array times six. Devices that you create with **10\_0 feature level** support only a single texture cube of six faces. Also, Direct3D 11 does not support partial cubemaps.

## 3D Textures

A 3D texture resource (also known as a volume texture) contains a 3D volume of texels. Because it is a texture resource, it may contain mipmap levels. 3D textures are represented by the [ID3D11Texture3D](#) interface. A fully populated 3D texture looks like the following illustration.



When a 3D texture mipmap slice is bound as a render target output (with a render-target view), the 3D texture behaves identically to a 2D texture array with n slices. The particular render slice is chosen from the geometry-shader stage, by declaring a scalar component of output data as the `SV_RenderTargetArrayIndex` system-value.

There is no concept of a 3D texture array; therefore a 3D texture subresource is a single mipmap level.

## Related topics

[Textures](#)

---

## Feedback

Was this page helpful?

Yes

No

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

# Texture Block Compression in Direct3D 11

Article • 08/19/2020

Block Compression (BC) support for textures has been extended in Direct3D 11 to include the BC6H and BC7 algorithms. BC6H supports high-dynamic range color source data, and BC7 provides better-than-average quality compression with less artifacts for standard RGB source data.

For more specific information about block compression algorithm support prior to Direct3D 11, including support for the BC1 through BC5 formats, see [Block Compression \(Direct3D 10\)](#).

**Note about File Formats:** The BC6H and BC7 texture compression formats use the DDS file format for storing the compressed texture data. For more information, see the [Programming Guide for DDS](#) for details.

## Block Compression Formats Supported in Direct3D 11

Source data	Minimum required data compression resolution	Recommended format	Minimum supported feature level
Three-channel color with alpha channel	Three color channels (5 bits:6 bits:5 bits), with 0 or 1 bit(s) of alpha	BC1	Direct3D 9.1
Three-channel color with alpha channel	Three color channels (5 bits:6 bits:5 bits), with 4 bits of alpha	BC2	Direct3D 9.1
Three-channel color with alpha channel	Three color channels (5 bits:6 bits:5 bits) with 8 bits of alpha	BC3	Direct3D 9.1
One-channel color	One color channel (8 bits)	BC4	Direct3D 10
Two-channel color	Two color channels (8 bits:8 bits)	BC5	Direct3D 10
Three-channel high dynamic range (HDR) color	Three color channels (16 bits:16 bits:16 bits) in "half" floating point*	BC6H	Direct3D 11
Three-channel color, alpha channel optional	Three color channels (4 to 7 bits per channel) with 0 to 8 bits of alpha	BC7	Direct3D 11

\*"Half" floating point is a 16 bit value that consists of an optional sign bit, a 5 bit biased exponent, and a 10 or 11 bit mantissa.

## BC1, BC2, and BC3 Formats

The BC1, BC2, and BC3 formats are equivalent to the Direct3D 9 DXTn texture compression formats, and are the same as the corresponding Direct3D 10 BC1, BC2, and BC3 formats. Support for these three formats is required by all feature levels (D3D\_FEATURE\_LEVEL\_9\_1, D3D\_FEATURE\_LEVEL\_9\_2, D3D\_FEATURE\_LEVEL\_9\_3, D3D\_FEATURE\_LEVEL\_10\_0, D3D\_FEATURE\_LEVEL\_10\_1, and D3D\_FEATURE\_LEVEL\_11\_0).

<b>Block compression format</b>	<b>DXGI format</b>	<b>Direct3D 9 equivalent format</b>	<b>Bytes per 4x4 pixel block</b>
BC1	DXGI_FORMAT_BC1_UNORM, DXGI_FORMAT_BC1_UNORM_SRGB, DXGI_FORMAT_BC1_TYPELESS	D3DFMT_DXT1, FourCC="DXT1"	8
BC2	DXGI_FORMAT_BC2_UNORM, DXGI_FORMAT_BC2_UNORM_SRGB, DXGI_FORMAT_BC2_TYPELESS	D3DFMT_DXT2*, FourCC="DXT2", D3DFMT_DXT3, FourCC="DXT3"	16
BC3	DXGI_FORMAT_BC3_UNORM, DXGI_FORMAT_BC3_UNORM_SRGB, DXGI_FORMAT_BC3_TYPELESS	D3DFMT_DXT4*, FourCC="DXT4", D3DFMT_DXT5, FourCC="DXT5"	16

\*These compression schemes (DXT2 and DXT4) make no distinction between the Direct3D 9 pre-multiplied alpha formats and the standard alpha formats. This distinction must be handled by the programmable shaders at render time.

## BC4 and BC5 Formats

<b>Block compression format</b>	<b>DXGI format</b>	<b>Direct3D 9 equivalent format</b>	<b>Bytes per 4x4 pixel block</b>
BC4	DXGI_FORMAT_BC4_UNORM, DXGI_FORMAT_BC4_SNORM, DXGI_FORMAT_BC4_TYPELESS	FourCC="ATI1"	8

<b>Block compression format</b>	<b>DXGI format</b>	<b>Direct3D 9 equivalent format</b>	<b>Bytes per 4x4 pixel block</b>
BC5	DXGI_FORMAT_BC5_UNORM, DXGI_FORMAT_BC5_SNORM, DXGI_FORMAT_BC5_TYPELESS	FourCC="ATI2"	16

## BC6H Format

For more detailed information about this format, see the [BC6H Format](#) documentation.

<b>Block compression format</b>	<b>DXGI format</b>	<b>Direct3D 9 equivalent format</b>	<b>Bytes per 4x4 pixel block</b>
BC6H	DXGI_FORMAT_BC6H_UF16, DXGI_FORMAT_BC6H_SF16, DXGI_FORMAT_BC6H_TYPELESS	N/A	16

The BC6H format can select different encoding modes for each 4x4 pixel block. A total of 14 different encoding modes are available, each with slightly different trade-offs in the resulting visual quality of the displayed texture. The choice of modes allows for fast decoding by the hardware with the quality level selected or adapted according to the source content, but it also greatly increases the complexity of the search space.

## BC7 Format

For more detailed information about this format, see the [BC7 Format](#) documentation.

<b>Block compression format</b>	<b>DXGI format</b>	<b>Direct3D 9 equivalent format</b>	<b>Bytes per 4x4 pixel block</b>
BC7	DXGI_FORMAT_BC7_UNORM, DXGI_FORMAT_BC7_UNORM_SRGB, DXGI_FORMAT_BC7_TYPELESS	N/A	16

The BC7 format can select different encoding modes for each 4x4 pixel block. A total of 8 different encoding modes are available, each with slightly different trade-offs in the resulting visual quality of the displayed texture. The choice of modes allows for fast decoding by the hardware with the quality level selected or adapted according to the source content, but it also greatly increases the complexity of the search space.

# In this section

Topic	Description
<a href="#">BC6H Format</a>	The BC6H format is a texture compression format designed to support high-dynamic range (HDR) color spaces in source data.
<a href="#">BC7 Format</a>	The BC7 format is a texture compression format used for high-quality compression of RGB and RGBA data.
<a href="#">BC7 Format Mode Reference</a>	This documentation contains a list of the 8 block modes and bit allocations for BC7 texture compression format blocks.

## Related topics

[Block Compression \(Direct3D 10\)](#)

[Textures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# BC6H Format

Article • 05/24/2021

The BC6H format is a texture compression format designed to support high-dynamic range (HDR) color spaces in source data.

- [About BC6H/DXGI\\_FORMAT\\_BC6H](#)
- [BC6H Implementation](#)
- [Decoding the BC6H Format](#)
- [BC6H Compressed Endpoint Format](#)
- [Sign Extension for Endpoint Values](#)
- [Transform Inversion for Endpoint Values](#)
- [Unquantization of Color Endpoints](#)
- [Related topics](#)

## About BC6H/DXGI\_FORMAT\_BC6H

The BC6H format provides high-quality compression for images that use three HDR color channels, with a 16-bit value for each color channel of the value (16:16:16). There is no support for an alpha channel.

BC6H is specified by the following DXGI\_FORMAT enumeration values:

- [DXGI\\_FORMAT\\_BC6H\\_TYPELESS](#).
- [DXGI\\_FORMAT\\_BC6H\\_UF16](#). This BC6H format does not use a sign bit in the 16-bit floating point color channel values.
- [DXGI\\_FORMAT\\_BC6H\\_SF16](#). This BC6H format uses a sign bit in the 16-bit floating point color channel values.

### Note

The 16 bit floating point format for color channels is often referred to as a "half" floating point format. This format has the following bit layout:

Format	Bit layout
UF16 (unsigned float)	5 exponent bits + 11 mantissa bits
SF16 (signed float)	1 sign bit + 5 exponent bits + 10 mantissa bits

The BC6H format can be used for [Texture2D](#) (including arrays), [Texture3D](#), or [TextureCube](#) (including arrays) texture resources. Similarly, this format applies to any MIP-map surfaces associated with these resources.

BC6H uses a fixed block size of 16 bytes (128 bits) and a fixed tile size of 4x4 texels. As with previous BC formats, texture images larger than the supported tile size (4x4) are compressed by using multiple blocks. This addressing identity applies also to three-dimensional images, MIP-maps, cubemaps, and texture arrays. All image tiles must be of the same format.

Some important notes about the BC6H format:

- BC6H supports floating point denormalization, but does not support INF (infinity) and NaN (not a number). The exception is the signed mode of BC6H (`DXGI_FORMAT_BC6H_SF16`), which supports -INF (negative infinity). Note that this support for -INF is merely an artifact of the format itself, and is not specifically supported by encoders for this format. In general, when encoders encounter INF (positive or negative) or NaN input data, they should convert that data to the maximum allowable non-INF representation value, and map NaN to 0 prior to compression.
- BC6H does not support an alpha channel.
- The BC6H decoder performs decompression before it performs texture filtering.
- BC6H decompression must be bit accurate; that is, the hardware must return results that are identical to the decoder described in this documentation.

## BC6H Implementation

A BC6H block consists of mode bits, compressed endpoints, compressed indices, and an optional partition index. This format specifies 14 different modes.

An endpoint color is stored as an RGB triplet. BC6H defines a palette of colors on an approximate line across a number of defined color endpoints. Also, depending on the mode, a tile can be divided into two regions or treated as a single region, where a two-region tile has a separate set of color endpoints for each region. BC6H stores one palette index per texel.

In the two-region case, there are 32 possible partitions.

## Decoding the BC6H Format

The pseudocode below shows the steps to decompress the pixel at (x,y) given the 16 byte BC6H block.

syntax

```

decompress_bc6h(x, y, block)
{
    mode = extract_mode(block);
    endpoints;
    index;

    if(mode.type == ONE)
    {
        endpoints = extract_compressed_endpoints(mode, block);
        index = extract_index_ONE(x, y, block);
    }
    else //mode.type == TWO
    {
        partition = extract_partition(block);
        region = get_region(partition, x, y);
        endpoints = extract_compressed_endpoints(mode, region, block);
        index = extract_index_TWO(x, y, partition, block);
    }

    unquantize(endpoints);
    color = interpolate(index, endpoints);
    finish_unquantize(color);
}

```

The following table contains the bit count and values for each of the 14 possible formats for BC6H blocks.

<b>Mode</b>	<b>Partition Indices</b>	<b>Partition</b>	<b>Color Endpoints</b>	<b>Mode Bits</b>
1	46 bits	5 bits	75 bits (10.555, 10.555, 10.555)	2 bits (00)
2	46 bits	5 bits	75 bits (7666, 7666, 7666)	2 bits (01)
3	46 bits	5 bits	72 bits (11.555, 11.444, 11.444)	5 bits (00010)
4	46 bits	5 bits	72 bits (11.444, 11.555, 11.444)	5 bits (00110)
5	46 bits	5 bits	72 bits (11.444, 11.444, 11.555)	5 bits (01010)
6	46 bits	5 bits	72 bits (9555, 9555, 9555)	5 bits (01110)
7	46 bits	5 bits	72 bits (8666, 8555, 8555)	5 bits (10010)
8	46 bits	5 bits	72 bits (8555, 8666, 8555)	5 bits (10110)
9	46 bits	5 bits	72 bits (8555, 8555, 8666)	5 bits (11010)

<b>Mode</b>	<b>Partition Indices</b>	<b>Partition</b>	<b>Color Endpoints</b>	<b>Mode Bits</b>
10	46 bits	5 bits	72 bits (6666, 6666, 6666)	5 bits (11110)
11	63 bits	0 bits	60 bits (10.10, 10.10, 10.10)	5 bits (00011)
12	63 bits	0 bits	60 bits (11.9, 11.9, 11.9)	5 bits (00111)
13	63 bits	0 bits	60 bits (12.8, 12.8, 12.8)	5 bits (01011)
14	63 bits	0 bits	60 bits (16.4, 16.4, 16.4)	5 bits (01111)

Each format in this table can be uniquely identified by the mode bits. The first ten modes are used for two-region tiles, and the mode bit field can be either two or five bits long. These blocks also have fields for the compressed color endpoints (72 or 75 bits), the partition (5 bits), and the partition indices (46 bits).

For the compressed color endpoints, the values in the preceding table note the precision of the stored RGB endpoints, and the number of bits used for each color value. For example, mode 3 specifies a color endpoint precision level of 11, and the number of bits used to store the delta values of the transformed endpoints for the red, blue and green colors (5, 4, and 4 respectively). Mode 10 does not use delta compression, and instead stores all four color endpoints explicitly.

The last four block modes are used for one-region tiles, where the mode field is 5 bits. These blocks have fields for the endpoints (60 bits) and the compressed indices (63 bits). Mode 11 (like Mode 10) does not use delta compression, and instead stores both color endpoints explicitly.

Modes 10011, 10111, 11011, and 11111 (not shown) are reserved. Do not use these in your encoder. If the hardware is passed blocks with one of these modes specified, the resulting decompressed block must contain all zeroes in all channels except for the alpha channel.

For BC6H, the alpha channel must always return 1.0 regardless of the mode.

## BC6H Partition Set

There are 32 possible partition sets for a two-region tile, and which are defined in the table below. Each 4x4 block represents a single shape.

0, 0, 1, 1,	0, 0, 0, 1,	0, 1, 1, 1,	0, 0, 0, 1,
0, 0, 1, 1,	0, 0, 0, 1,	0, 1, 1, 1,	0, 0, 1, 1,
0, 0, 1, 1,	0, 0, 0, 1,	0, 1, 1, 1,	0, 0, 1, 1,
0, 0, 1, 1,	0, 0, 0, 1,	0, 1, 1, 1,	0, 1, 1, 1,
0, 0, 0, 0,	0, 0, 1, 1,	0, 0, 0, 1,	0, 0, 0, 0,
0, 0, 0, 1,	0, 1, 1, 1,	0, 0, 1, 1,	0, 0, 0, 1,
0, 0, 0, 1,	0, 1, 1, 1,	0, 1, 1, 1,	0, 0, 1, 1,
0, 0, 1, 1,	1, 1, 1, 1,	1, 1, 1, 1,	0, 1, 1, 1,
0, 0, 0, 0,	0, 0, 1, 1,	0, 0, 0, 0,	0, 0, 0, 0,
0, 0, 0, 0,	0, 1, 1, 1,	0, 0, 0, 1,	0, 0, 0, 0,
0, 0, 0, 1,	1, 1, 1, 1,	0, 1, 1, 1,	0, 0, 0, 1,
0, 0, 1, 1,	1, 1, 1, 1,	1, 1, 1, 1,	0, 1, 1, 1,
0, 0, 0, 1,	0, 0, 0, 0,	0, 0, 0, 0,	0, 0, 0, 0,
0, 1, 1, 1,	0, 0, 0, 0,	1, 1, 1, 1,	0, 0, 0, 0,
1, 1, 1, 1,	1, 1, 1, 1,	1, 1, 1, 1,	0, 0, 0, 0,
1, 1, 1, 1,	1, 1, 1, 1,	1, 1, 1, 1,	1, 1, 1, 1,
0, 0, 0, 0,	0, 1, 1, 1,	0, 0, 0, 0,	0, 1, 1, 1,
1, 0, 0, 0,	0, 0, 0, 1,	0, 0, 0, 0,	0, 0, 1, 1,
1, 1, 1, 0,	0, 0, 0, 0,	1, 0, 0, 0,	0, 0, 0, 1,
1, 1, 1, 1,	0, 0, 0, 0,	1, 1, 1, 0,	0, 0, 0, 0,
0, 0, 1, 1,	0, 0, 0, 0,	0, 0, 0, 0,	0, 1, 1, 1,
0, 0, 0, 1,	1, 0, 0, 0,	0, 0, 0, 0,	0, 0, 1, 1,
0, 0, 0, 0,	1, 1, 0, 0,	1, 0, 0, 0,	0, 0, 1, 1,
0, 0, 0, 0,	1, 1, 1, 0,	1, 1, 0, 0,	0, 0, 0, 1,
0, 0, 1, 1,	0, 0, 0, 0,	0, 1, 1, 0,	0, 0, 1, 1,
0, 0, 0, 1,	1, 0, 0, 0,	0, 1, 1, 0,	0, 1, 1, 0,
0, 0, 0, 0,	1, 1, 0, 0,	0, 1, 1, 0,	1, 1, 0, 0,
0, 0, 1, 1,	0, 0, 0, 0,	0, 1, 1, 1,	0, 0, 1, 1,
0, 1, 1, 1,	1, 1, 1, 1,	0, 0, 0, 1,	1, 0, 0, 1,
1, 1, 1, 0,	1, 1, 1, 1,	1, 0, 0, 0,	1, 0, 0, 1,
1, 0, 0, 0,	0, 0, 0, 0,	1, 1, 1, 0,	1, 1, 0, 0,

In this table of partition sets, the bolded and underlined entry is the location of the fix-up index for subset 1 (which is specified with one less bit). The fix-up index for subset 0 is always index 0, as the partitioning is always arranged such that index 0 is always in subset 0. Partition order goes from top-left to bottom-right, moving left to right and then top to bottom.

## BC6H Compressed Endpoint Format

Header Bit	10 5 5 5	7 6 6 6	11 5 4 4	11 4 5 4	11 4 4 5	9 5 5 5	8 6 5 5	8 5 6 5	8 5 5 6	6 6 6 6	10 10	11 9	12 8	16 4
0														
1	m[1:0]	m[1:0]												
2	gy[4]	gy[5]												
3	by[4]	gz[4]												
4	bz[4]	gz[5]												
5														
6														
7														
8														
9														
10														
11		rw[6:0]												
12		bz[0]												
13		bz[1]												
14	rw[9:0]	by[4]												
15														
16														
17														
18														
19														
20														
21		gw[6:0]												
22		by[5]												
23		bz[2]												
24	gw[9:0]	gy[4]												
25														
26														
27														
28														
29														
30														
31		bw[6:0]												
32		bz[3]												
33		bz[5]												
34	bw[9:0]	bz[4]												
35														
36														
37														
38	rx[4:0]		rx[3:0]	rx[3:0]										rx[3:0]
39	gx[4:0]		rx[4:0]	rw[10]	rw[10]	rx[4:0]								
40	gz[4]		rx[5:0]	rw[10]	gz[4]	by[4]	gz[4]	rx[5:0]	rx[4:0]	rx[4:0]	rx[5:0]			
41														
42														rx[7:0]
43														
44	gy[3:0]	gy[3:0]	gy[3:0]	gy[3:0]	gy[3:0]	gy[3:0]	gy[3:0]	gy[3:0]	gy[3:0]	gy[3:0]	rx[8:0]	rx[10:11]	rw[10:15]	
45														
46														
47														
48	gx[4:0]		gx[3:0]	gw[10]	gx[4:0]	gw[10]	gx[4:0]	gx[4:0]	gx[4:0]	gx[4:0]				gx[3:0]
49	bz[0]		gx[5:0]	bz[0]	gw[10]	bz[0]	bz[0]	bz[0]	bz[0]	bz[0]				
50														
51														
52														
53														
54	gz[3:0]	gz[3:0]	gz[3:0]	gz[3:0]	gz[3:0]	gz[3:0]	gz[3:0]	gz[3:0]	gz[3:0]	gz[3:0]	gx[8:0]	gw[10:11]	gw[10:15]	
55														
56														
57														
58	bx[4:0]		bx[3:0]	bx[3:0]		bx[4:0]	bx[4:0]	bx[4:0]	bx[4:0]	bx[4:0]				bx[3:0]
59	bz[1]		bx[5:0]	bz[1]	bw[10]	bw[10]	bz[1]	bz[1]	bz[1]	bz[1]				
60														
61														
62														
63														
64	by[3:0]	by[3:0]	by[3:0]	by[3:0]	by[3:0]	by[3:0]	by[3:0]	by[3:0]	by[3:0]	by[3:0]	bx[8:0]	bx[10:11]	bw[10:15]	
65														
66														
67														
68	ry[4:0]		ry[4:0]	bz[0]	bz[1]	bz[2]	ry[4:0]		ry[4:0]	ry[4:0]				
69	bz[2]		ry[5:0]	bz[2]	bz[2]	bz[2]	bz[2]	ry[5:0]	bz[2]	bz[2]	ry[5:0]			
70														
71														
72														
73														
74	rz[4:0]		rz[4:0]	gy[4]	bz[4]	bz[3]	rz[4:0]		rz[4:0]	rz[4:0]				
75	bz[3]		rz[5:0]	bz[3]	bz[3]	bz[3]	bz[3]	rz[5:0]	bz[3]	bz[3]	rz[5:0]			
76														
77														
78														
79														
80														
81	d[4:0]	d[4:0]	d[4:0]	d[4:0]	d[4:0]	d[4:0]	d[4:0]	d[4:0]	d[4:0]	d[4:0]	10 10	11 9	12 8	16 4
	10 5 5 5	7 6 6 6	11 5 4 4	11 4 5 4	11 4 4 5	9 5 5 5	8 6 5 5	8 5 6 5	8 5 5 6	6 6 6 6				

This table shows the bit fields for the compressed endpoints as a function of the endpoint format, with each column specifying an encoding and each row specifying a bit field. This approach takes up 82 bits for two-region tiles and 65 bits for one-region

tiles. As an example, the first 5 bits for the one-region [16 4] encoding above (specifically the right-most column) are bits m[4:0], the next 10 bits are bits rw[9:0], and so on with the last 6 bits containing bw[10:15].

The field names in the table above are defined as follows:

<b>Field</b>	<b>Variable</b>
m	mode
d	shape index
rw	endpt[0].A[0]
rx	endpt[0].B[0]
ry	endpt[1].A[0]
rz	endpt[1].B[0]
gw	endpt[0].A[1]
gx	endpt[0].B[1]
gy	endpt[1].A[1]
gz	endpt[1].B[1]
bw	endpt[0].A[2]
bx	endpt[0].B[2]
by	endpt[1].A[2]
bz	endpt[1].B[2]

Endpt[i], where i is either 0 or 1, refers to the 0th or 1st set of endpoints respectively.

## Sign Extension for Endpoint Values

For two-region tiles, there are four endpoint values that can be sign extended.

Endpt[0].A is signed only if the format is a signed format; the other endpoints are signed only if the endpoint was transformed, or if the format is a signed format. The code below demonstrates the algorithm for extending the sign of two-region endpoint values.

```

static void sign_extend_two_region(Pattern &p, IntEndpts
endpts[NREGIONS_TWO])
{
    for (int i=0; i<NCHANNELS; ++i)
    {
        if (BC6H::FORMAT == SIGNED_F16)
            endpts[0].A[i] = SIGN_EXTEND(endpts[0].A[i], p.chan[i].prec);
        if (p.transformed || BC6H::FORMAT == SIGNED_F16)
        {
            endpts[0].B[i] = SIGN_EXTEND(endpts[0].B[i], p.chan[i].delta[0]);
            endpts[1].A[i] = SIGN_EXTEND(endpts[1].A[i], p.chan[i].delta[1]);
            endpts[1].B[i] = SIGN_EXTEND(endpts[1].B[i], p.chan[i].delta[2]);
        }
    }
}

```

For one-region tiles, the behavior is the same, only with endpt[1] removed.

#### syntax

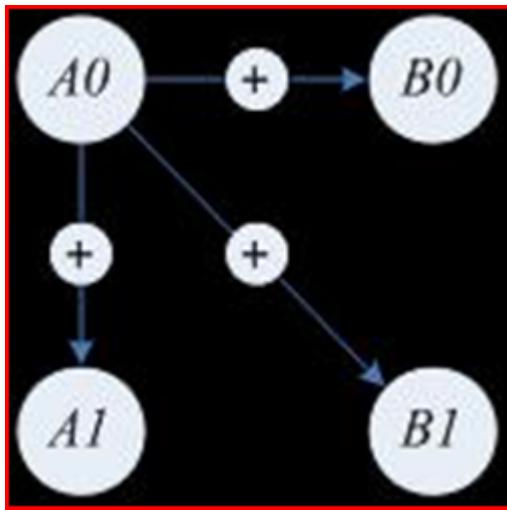
```

static void sign_extend_one_region(Pattern &p, IntEndpts
endpts[NREGIONS_ONE])
{
    for (int i=0; i<NCHANNELS; ++i)
    {
        if (BC6H::FORMAT == SIGNED_F16)
            endpts[0].A[i] = SIGN_EXTEND(endpts[0].A[i], p.chan[i].prec);
        if (p.transformed || BC6H::FORMAT == SIGNED_F16)
            endpts[0].B[i] = SIGN_EXTEND(endpts[0].B[i], p.chan[i].delta[0]);
    }
}

```

## Transform Inversion for Endpoint Values

For two-region tiles, the transform applies the inverse of the difference encoding, adding the base value at endpt[0].A to the three other entries for a total of 9 add operations. In the image below, the base value is represented as "A0" and has the highest floating point precision. "A1," "B0," and "B1" are all deltas calculated from the anchor value, and these delta values are represented with lower precision. (A0 corresponds to endpt[0].A, B0 corresponds to endpt[0].B, A1 corresponds to endpt[1].A, and B1 corresponds to endpt[1].B.)



For one-region tiles there is only one delta offset, and therefore only 3 add operations.

The decompressor must ensure that the results of the inverse transform will not overflow the precision of `endpt[0].a`. In the case of an overflow, the values resulting from the inverse transform must wrap within the same number of bits. If the precision of `A0` is "p" bits, then the transform algorithm is:

$$B0 = (B0 + A0) \& ((1 \ll p) - 1)$$

For signed formats, the results of the delta calculation must be sign extended as well. If the sign extension operation considers extending both signs, where 0 is positive and 1 is negative, then the sign extension of 0 takes care of the clamp above. Equivalently, after the clamp above, only a value of 1 (negative) needs to be sign extended.

## Unquantization of Color Endpoints

Given the uncompressed endpoints, the next step is to perform an initial unquantization of the color endpoints. This involves three steps:

- An unquantization of the color palettes
- Interpolation of the palettes
- Unquantization finalization

Separating the unquantization process into two parts (color palette unquantization before interpolation and final unquantization after interpolation) reduces the number of multiplication operations required when compared to a full unquantization process before palette interpolation.

The code below illustrates the process for retrieving estimates of the original 16-bit color values, and then using the supplied weight values to add 6 additional color values to the palette. The same operation is performed on each channel.

syntax

```
int aWeight3[] = {0, 9, 18, 27, 37, 46, 55, 64};  
int aWeight4[] = {0, 4, 9, 13, 17, 21, 26, 30, 34, 38, 43, 47, 51, 55, 60,  
64};  
  
// c1, c2: endpoints of a component  
void generate_palette_unquantized(UINT8 uNumIndices, int c1, int c2, int  
prec, UINT16 palette[NINDICES])  
{  
    int* aWeights;  
    if(uNumIndices == 8)  
        aWeights = aWeight3;  
    else // uNumIndices == 16  
        aWeights = aWeight4;  
  
    int a = unquantize(c1, prec);  
    int b = unquantize(c2, prec);  
  
    // interpolate  
    for(int i = 0; i < uNumIndices; ++i)  
        palette[i] = finish_unquantize((a * (64 - aWeights[i]) + b *  
aWeights[i] + 32) >> 6);  
}
```

The next code sample demonstrates the interpolation process, with the following observations:

- Since the full range of color values for the **unquantize** function (below) are from -32768 to 65535, the interpolator is implemented using 17-bit signed arithmetic.
- After interpolation, the values are passed to the **finish\_unquantize** function (described in the third sample in this section), which applies the final scaling.
- All hardware decompressors are required to return bit-accurate results with these functions.

syntax

```
int unquantize(int comp, int uBitsPerComp)  
{  
    int unq, s = 0;  
    switch(BC6H::FORMAT)  
    {  
        case UNSIGNED_F16:  
            if(uBitsPerComp >= 15)  
                unq = comp;  
            else if(comp == 0)  
                unq = 0;  
            else if(comp == ((1 << uBitsPerComp) - 1))  
                unq = 0xFFFF;  
            else  
                unq = ((comp << 16) + 0x8000) >> uBitsPerComp;
```

```

        break;

    case SIGNED_F16:
        if(uBitsPerComp >= 16)
            unq = comp;
        else
        {
            if(comp < 0)
            {
                s = 1;
                comp = -comp;
            }

            if(comp == 0)
                unq = 0;
            else if(comp >= ((1 << (uBitsPerComp - 1)) - 1))
                unq = 0x7FFF;
            else
                unq = ((comp << 15) + 0x4000) >> (uBitsPerComp-1);

            if(s)
                unq = -unq;
        }
        break;
    }
    return unq;
}

```

**finish\_unquantize** is called after palette interpolation. The **unquantize** function postpones the scaling by 31/32 for signed, 31/64 for unsigned. This behavior is required to get the final value into valid half range(-0x7BFF ~ 0x7BFF) after the palette interpolation is completed in order to reduce the number of necessary multiplications. **finish\_unquantize** applies the final scaling and returns an **unsigned short** value that gets reinterpreted into **half**.

#### syntax

```

unsigned short finish_unquantize(int comp)
{
    if(BC6H::FORMAT == UNSIGNED_F16)
    {
        comp = (comp * 31) >> 6;                                // scale the magnitude by 31/64
        return (unsigned short) comp;
    }
    else // (BC6H::FORMAT == SIGNED_F16)
    {
        comp = (comp < 0) ? -((( -comp) * 31) >> 5) : (comp * 31) >> 5; // scale the magnitude by 31/32
        int s = 0;
        if(comp < 0)

```

```
{  
    s = 0x8000;  
    comp = -comp;  
}  
return (unsigned short) (s | comp);  
}  
}
```

## Related topics

[Texture Block Compression in Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# BC7 Format

Article • 12/14/2022

The BC7 format is a texture compression format used for high-quality compression of RGB and RGBA data.

- [About BC7/DXGI\\_FORMAT\\_BC7](#)
- [BC7 Implementation](#)
- [Decoding the BC7 Format](#)
- [Related topics](#)

For info about the block modes of the BC7 format, see [BC7 Format Mode Reference](#).

## About BC7/DXGI\_FORMAT\_BC7

BC7 is specified by the following DXGI\_FORMAT enumeration values:

- [DXGI\\_FORMAT\\_BC7\\_TYPELESS](#).
- [DXGI\\_FORMAT\\_BC7\\_UNORM](#).
- [DXGI\\_FORMAT\\_BC7\\_UNORM\\_SRGB](#).

The BC7 format can be used for [Texture2D](#) (including arrays), [Texture3D](#), or [TextureCube](#) (including arrays) texture resources. Similarly, this format applies to any MIP-map surfaces associated with these resources.

BC7 uses a fixed block size of 16 bytes (128 bits) and a fixed tile size of 4x4 texels. As with previous BC formats, texture images larger than the supported tile size (4x4) are compressed by using multiple blocks. This addressing identity also applies to three-dimensional images and MIP-maps, cubemaps, and texture arrays. All image tiles must be of the same format.

BC7 compresses both three-channel (RGB) and four-channel (RGBA) fixed-point data images. Typically, source data is 8 bits per color component (channel), although the format is capable of encoding source data with higher bits per color component. All image tiles must be of the same format.

The BC7 decoder performs decompression before texture filtering is applied.

BC7 decompression hardware must be bit accurate; that is, the hardware must return results that are identical to the results returned by the decoder described in this document.

# BC7 Implementation

A BC7 implementation can specify one of 8 modes, with the mode specified in the least significant bit of the 16 byte (128 bit) block. The mode is encoded by zero or more bits with a value of 0 followed by a 1.

A BC7 block can contain multiple endpoint pairs. For the purposes of this documentation, the set of indices that correspond to an endpoint pair may be referred to as a "subset." Also, in some block modes, the endpoint representation is encoded in a form that -- again, for the purposes of this documentation -- shall be referred to as "RGBP," where the "P" bit represents a shared least significant bit for the color components of the endpoint. For example, if the endpoint representation for the format is "RGB 5.5.5.1," then the endpoint is interpreted as an RGB 6.6.6 value, where the state of the P-bit defines the least significant bit of each component. Similarly, for source data with an alpha channel, if the representation for the format is "RGBAP 5.5.5.5.1," then the endpoint is interpreted as RGBA 6.6.6.6. Depending on the block mode, you can specify the shared least significant bit for either both endpoints of a subset individually (2 P-bits per subset), or shared between endpoints of a subset (1 P-bit per subset).

For BC7 blocks that don't explicitly encode the alpha component, a BC7 block consists of mode bits, partition bits, compressed endpoints, compressed indices, and an optional P-bit. In these blocks the endpoints have an RGB-only representation and the alpha component is decoded as 1.0 for all texels in the source data.

For BC7 blocks that have combined color and alpha components, a block consists of mode bits, compressed endpoints, compressed indices, and optional partition bits and a P-bit. In these blocks, the endpoint colors are expressed in RGBA format, and alpha component values are interpolated alongside the color component values.

For BC7 blocks that have separate color and alpha components, a block consists of mode bits, rotation bits, compressed endpoints, compressed indices, and an optional index selector bit. These blocks have an effective RGB vector [R, G, B] and a scalar alpha channel [A] separately encoded.

The following table lists the components of each block type.

<b>BC7 block contains...</b>	<b>mode bits</b>	<b>rotation bits</b>	<b>index selector bit</b>	<b>partition bits</b>	<b>compressed endpoints</b>	<b>P-bit</b>	<b>compressed indices</b>
color components only	required	N/A	N/A	required	required	optional	required

<b>BC7 block contains...</b>	<b>mode bits</b>	<b>rotation bits</b>	<b>index selector bit</b>	<b>partition bits</b>	<b>compressed endpoints</b>	<b>P-bit</b>	<b>compressed indices</b>
color + alpha combined	required	N/A	N/A	optional	required	optional	required
color and alpha separated	required	required	optional	N/A	required	N/A	required

BC7 defines a palette of colors on an approximate line between two endpoints. The mode value determines the number of interpolating endpoint pairs per block. BC7 stores one palette index per texel.

For each subset of indices that corresponds to a pair of endpoints, the encoder fixes the state of one bit of the compressed index data for that subset. It does so by choosing an endpoint order that allows the index for the designated "fix-up" index to set its most significant bit to 0, and which can then be discarded, saving one bit per subset. For block modes with only a single subset, the fix-up index is always index 0.

## Decoding the BC7 Format

The following pseudocode outlines the steps to decompress the pixel at (x,y) given the 16 byte BC7 block.

### syntax

```

decompress_bc7(x, y, block)
{
    mode = extract_mode(block);

    //decode partition data from explicit partition bits
    subset_index = 0;
    num_subsets = 1;

    if (mode.type == 0 OR == 1 OR == 2 OR == 3 OR == 7)
    {
        num_subsets = get_num_subsets(mode.type);
        partition_set_id = extract_partition_set_id(mode, block);
        subset_index = get_partition_index(num_subsets, partition_set_id, x,
y);
    }

    //extract raw, compressed endpoint bits

```

```

    UINT8 endpoint_array[2 * num_subsets][4] = extract_endpoints(mode,
block);

    //decode endpoint color and alpha for each subset
    fully_decode_endpoints(endpoint_array, mode, block);

    //endpoints are now complete.
    UINT8 endpoint_start[4] = endpoint_array[2 * subset_index];
    UINT8 endpoint_end[4]   = endpoint_array[2 * subset_index + 1];

    //Determine the palette index for this pixel
    alpha_index      = get_alpha_index(block, mode, x, y);
    alpha_bitcount   = get_alpha_bitcount(block, mode);
    color_index      = get_color_index(block, mode, x, y);
    color_bitcount   = get_color_bitcount(block, mode);

    //determine output
    UINT8 output[4];
    output.rgb = interpolate(endpoint_start.rgb, endpoint_end.rgb,
color_index, color_bitcount);
    output.a   = interpolate(endpoint_start.a,   endpoint_end.a,
alpha_index, alpha_bitcount);

    if (mode.type == 4 OR == 5)
    {
        //Decode the 2 color rotation bits as follows:
        // 00 - Block format is Scalar(A) Vector(RGB) - no swapping
        // 01 - Block format is Scalar(R) Vector(AGB) - swap A and R
        // 10 - Block format is Scalar(G) Vector(RAB) - swap A and G
        // 11 - Block format is Scalar(B) Vector(RGA) - swap A and B
        rotation = extract_rot_bits(mode, block);
        output = swap_channels(output, rotation);
    }

}

```

The following pseudocode outlines the steps to fully decode endpoint color and alpha components for each subset given a 16-byte BC7 block.

#### syntax

```

fully_decode_endpoints(endpoint_array, mode, block)
{
    //first handle modes that have P-bits
    if (mode.type == 0 OR == 1 OR == 3 OR == 6 OR == 7)
    {
        for each endpoint i
        {
            //component-wise left-shift
            endpoint_array[i].rgba = endpoint_array[i].rgba << 1;
        }

        //if P-bit is shared
    }
}

```

```

        if (mode.type == 1)
        {
            pbit_zero = extract_pbit_zero(mode, block);
            pbit_one = extract_pbit_one(mode, block);

            //rgb component-wise insert pbits
            endpoint_array[0].rgb |= pbit_zero;
            endpoint_array[1].rgb |= pbit_zero;
            endpoint_array[2].rgb |= pbit_one;
            endpoint_array[3].rgb |= pbit_one;
        }
        else //unique P-bit per endpoint
        {
            pbit_array = extract_pbit_array(mode, block);
            for each endpoint i
            {
                endpoint_array[i].rgba |= pbit_array[i];
            }
        }
    }

    for each endpoint i
    {
        // Color_component_precision & alpha_component_precision includes
        pbit
        // left shift endpoint components so that their MSB lies in bit 7
        endpoint_array[i].rgb = endpoint_array[i].rgb << (8 -
color_component_precision(mode));
        endpoint_array[i].a = endpoint_array[i].a << (8 -
alpha_component_precision(mode));

        // Replicate each component's MSB into the LSBs revealed by the
        // left-shift operation above
        endpoint_array[i].rgb = endpoint_array[i].rgb |
(endpoint_array[i].rgb >> color_component_precision(mode));
        endpoint_array[i].a = endpoint_array[i].a | (endpoint_array[i].a >>
alpha_component_precision(mode));
    }

    //If this mode does not explicitly define the alpha component
    //set alpha equal to 1.0
    if (mode.type == 0 OR == 1 OR == 2 OR == 3)
    {
        for each endpoint i
        {
            endpoint_array[i].a = 255; //i.e. alpha = 1.0f
        }
    }
}

```

To generate each interpolated component for each subset, use the following algorithm:  
let "c" be the component to generate; let "e0" be that component of endpoint 0 of the  
subset; and let "e1" be that component of endpoint 1 of the subset.

#### syntax

```
UINT16 aWeights2[] = {0, 21, 43, 64};  
UINT16 aWeights3[] = {0, 9, 18, 27, 37, 46, 55, 64};  
UINT16 aWeights4[] = {0, 4, 9, 13, 17, 21, 26, 30, 34, 38, 43, 47, 51, 55,  
60, 64};  
  
UINT8 interpolate(UINT8 e0, UINT8 e1, UINT8 index, UINT8 indexprecision)  
{  
    if(indexprecision == 2)  
        return (UINT8) (((64 - aWeights2[index])*UINT16(e0) +  
aWeights2[index]*UINT16(e1) + 32) >> 6);  
    else if(indexprecision == 3)  
        return (UINT8) (((64 - aWeights3[index])*UINT16(e0) +  
aWeights3[index]*UINT16(e1) + 32) >> 6);  
    else // indexprecision == 4  
        return (UINT8) (((64 - aWeights4[index])*UINT16(e0) +  
aWeights4[index]*UINT16(e1) + 32) >> 6);  
}
```

The following pseudocode illustrates how to extract indices and bit counts for color and alpha components. Blocks with separate color and alpha also have two sets of index data: one for the vector channel, and one for the scalar channel. For Mode 4, these indices are of differing widths (2 or 3 bits), and there is a one-bit selector which specifies whether the vector or scalar data uses the 3-bit indices. (Extracting the alpha bit count is similar to extracting color bit count but with inverse behavior based on the **idxMode** bit.)

#### syntax

```
bitcount get_color_bitcount(block, mode)  
{  
    if (mode.type == 0 OR == 1)  
        return 3;  
  
    if (mode.type == 2 OR == 3 OR == 5 OR == 7)  
        return 2;  
  
    if (mode.type == 6)  
        return 4;  
  
    //The only remaining case is Mode 4 with 1-bit index selector  
    idxMode = extract_idxMode(block);  
    if (idxMode == 0)  
        return 2;  
    else  
        return 3;  
}
```

## Related topics

[Texture Block Compression in Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# BC7 Format Mode Reference

Article • 07/20/2021

This documentation contains a list of the 8 block modes and bit allocations for BC7 texture compression format blocks.

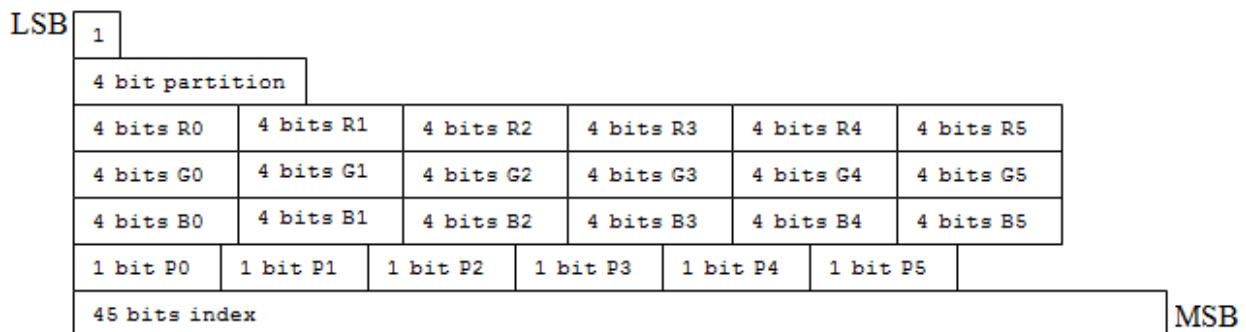
The colors for each subset within a block are represented by two explicit endpoint colors and a set of interpolated colors between them. Depending on the block's index precision, each subset can have 4, 8 or 16 possible colors.

- [Mode 0](#)
- [Mode 1](#)
- [Mode 2](#)
- [Mode 3](#)
- [Mode 4](#)
- [Mode 5](#)
- [Mode 6](#)
- [Mode 7](#)
- [Remarks](#)
- [Related topics](#)

## Mode 0

BC7 Mode 0 has the following characteristics:

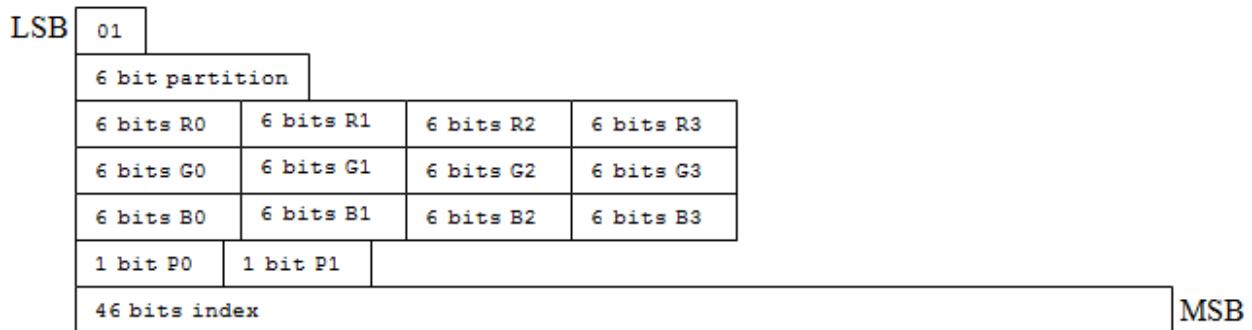
- Color components only (no alpha)
- 3 subsets per block
- RGBP 4.4.4.1 endpoints with a unique P-bit per endpoint
- 3-bit indices
- 16 partitions



## Mode 1

BC7 Mode 1 has the following characteristics:

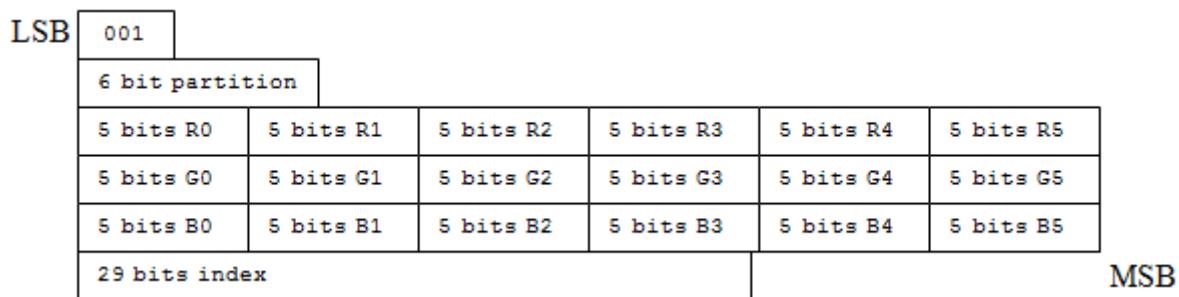
- Color components only (no alpha)
- 2 subsets per block
- RGBP 6.6.6.1 endpoints with a shared P-bit per subset)
- 3-bit indices
- 64 partitions



## Mode 2

BC7 Mode 2 has the following characteristics:

- Color components only (no alpha)
- 3 subsets per block
- RGB 5.5.5 endpoints
- 2-bit indices
- 64 partitions

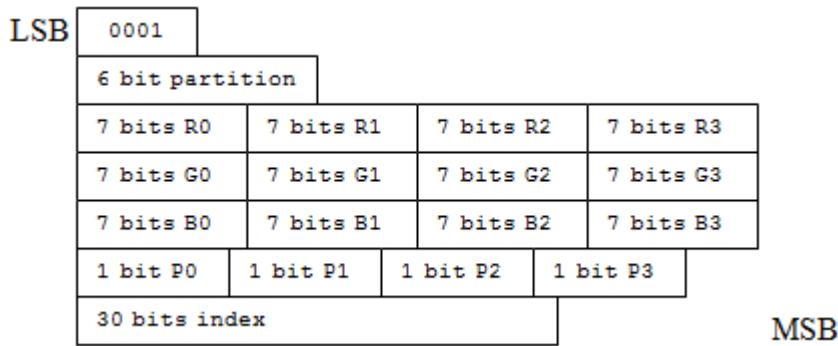


## Mode 3

BC7 Mode 3 has the following characteristics:

- Color components only (no alpha)
- 2 subsets per block
- RGBP 7.7.7.1 endpoints with a unique P-bit per subset)
- 2-bit indices

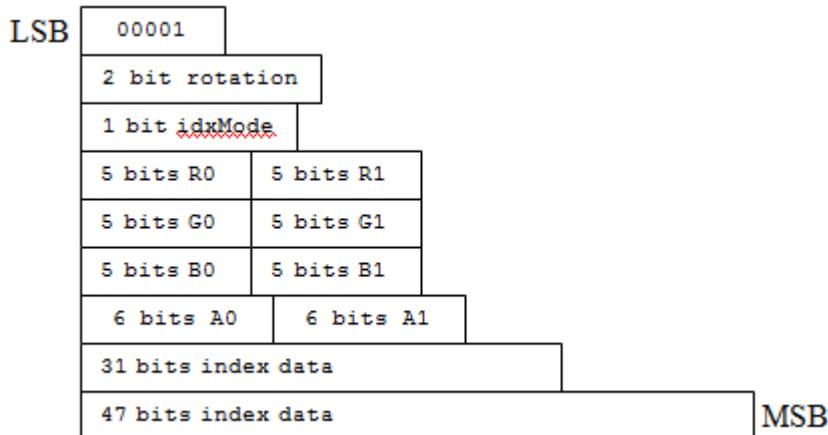
- 64 partitions



## Mode 4

BC7 Mode 4 has the following characteristics:

- Color components with separate alpha component
- 1 subset per block
- RGB 5.5.5 color endpoints
- 6-bit alpha endpoints
- 16 x 2-bit indices
- 16 x 3-bit indices
- 2-bit component rotation
- 1-bit index selector (whether the 2- or 3-bit indices are used)

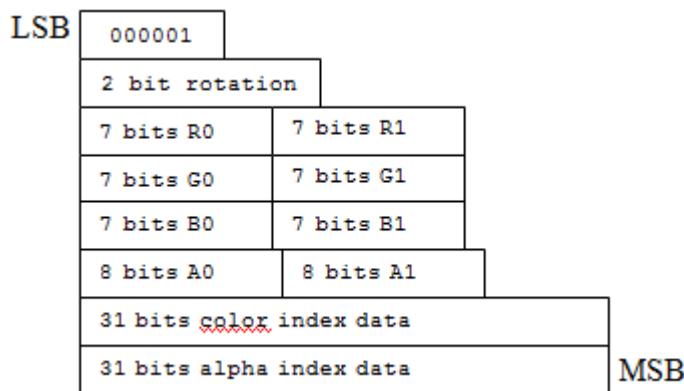


## Mode 5

BC7 Mode 5 has the following characteristics:

- Color components with separate alpha component
- 1 subset per block
- RGB 7.7.7 color endpoints
- 8-bit alpha endpoints

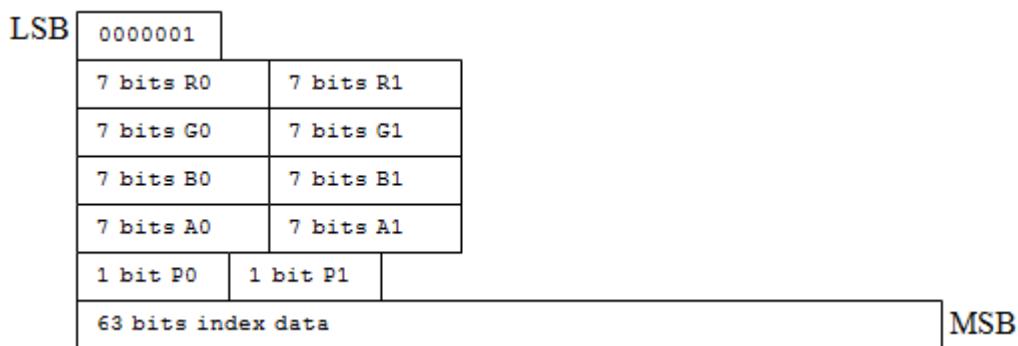
- 16 x 2-bit color indices
- 16 x 2-bit alpha indices
- 2-bit component rotation



## Mode 6

BC7 Mode 6 has the following characteristics:

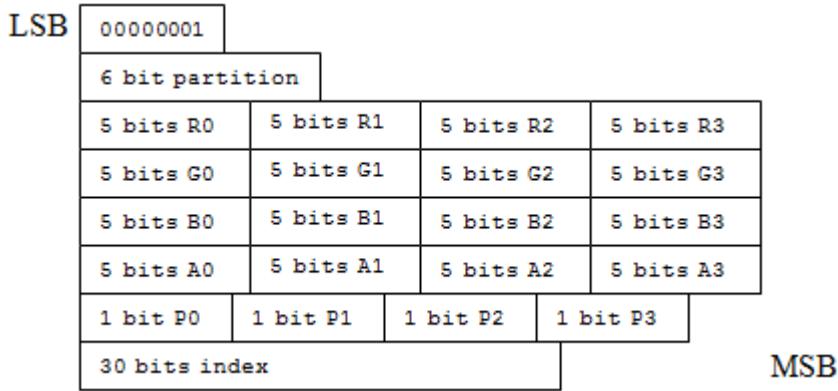
- Combined color and alpha components
- One subset per block
- RGBAP 7.7.7.7.1 color (and alpha) endpoints (unique P-bit per endpoint)
- 16 x 4-bit indices



## Mode 7

BC7 Mode 7 has the following characteristics:

- Combined color and alpha components
- 2 subsets per block
- RGBAP 5.5.5.5.1 color (and alpha) endpoints (unique P-bit per endpoint)
- 2-bit indices
- 64 partitions



## Remarks

Mode 8 (the least significant byte is set to 0x00) is reserved. Don't use it in your encoder. If you pass this mode to the hardware, a block initialized to all zeroes is returned.

In BC7, you can encode the alpha component in one of the following ways:

- Block types without explicit alpha component encoding. In these blocks, the color endpoints have an RGB-only encoding, with the alpha component decoded to 1.0 for all texels.
- Block types with combined color and alpha components. In these blocks, the endpoint color values are specified in the RGBA format, and the alpha component values are interpolated along with the color values.
- Block types with separated color and alpha components. In these blocks the color and alpha values are specified separately, each with their own set of indices. As a result, they have an effective vector and a scalar channel separately encoded, where the vector commonly specifies the color channels [R, G, B] and the scalar specifies the alpha channel [A]. To support this approach, a separate 2-bit field is provided in the encoding, which permits the specification of the separate channel encoding as a scalar value. As a result, the block can have one of the following four different representations of this alpha encoding (as indicated by the 2-bit field):
  - RGB|A: alpha channel separate
  - AGB|R: "red" color channel separate
  - RAB|G: "green" color channel separate
  - RGA|B: "blue" color channel separate

The decoder reorders the channel order back to RGBA after decoding, so the internal block format is invisible to the developer. Blocks with separate color and alpha components also have two sets of index data: one for the vectored set of channels, and one for the scalar channel. (In the case of Mode 4, these indices are

of differing widths [2 or 3 bits]. Mode 4 also contains a 1-bit selector that specifies whether the vector or the scalar channel uses the 3-bit indices.)

## Related topics

[BC7 Format](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Floating-point rules (Direct3D 11)

Article • 02/08/2025

Direct3D 11 supports several floating-point representations. All floating-point computations operate under a defined subset of the IEEE 754 32-bit single precision floating-point rules.

- 32-bit floating-point rules
  - Honored IEEE-754 rules
  - Deviations or additional requirements from IEEE-754 rules
- 64-bit (double precision) floating point rules
- 16-bit floating-point rules
- 11-bit and 10-bit floating-point rules
- Related topics

## 32-bit floating-point rules

There are two sets of rules: those that conform to IEEE-754, and those that deviate from the standard.

### Honored IEEE-754 rules

Some of these rules are a single option where IEEE-754 offers choices.

- Divide by 0 produces +/- INF, except 0/0 which results in NaN.
- log of (+/-) 0 produces -INF. log of a negative value (other than -0) produces NaN.
- Reciprocal square root (rsq) or square root (sqrt) of a negative number produces NaN. The exception is -0; sqrt(-0) produces -0, and rsq(-0) produces -INF.
- INF - INF = NaN
- (+/-)INF / (+/-)INF = NaN
- (+/-)INF \* 0 = NaN
- NaN (any OP) any-value = NaN
- The comparisons EQ, GT, GE, LT, and LE, when either or both operands is NaN returns FALSE.
- Comparisons ignore the sign of 0 (so +0 equals -0).
- The comparison NE, when either or both operands is NaN returns TRUE.
- Comparisons of any non-NaN value against +/- INF return the correct result.

## Deviations or additional requirements from IEEE-754 rules

- IEEE-754 requires floating-point operations to produce a result that is the nearest representable value to an infinitely-precise result, known as round-to-nearest-even. Direct3D 11 defines the same requirement: 32-bit floating-point operations produce a result that is within 0.5 unit-last-place (ULP) of the infinitely-precise result. This means that, for example, hardware is allowed to truncate results to 32-bit rather than perform round-to-nearest-even, as that would result in error of at most 0.5 ULP. This rule applies only to addition, subtraction, and multiplication.
- There is no support for floating-point exceptions, status bits or traps.
- Denorms are flushed to sign-preserved zero on input and output of any floating-point mathematical operation. Exceptions are made for any I/O or data movement operation that doesn't manipulate the data.
- States that contain floating-point values, such as Viewport MinDepth/MaxDepth, BorderColor values, may be provided as denorm values and may or may not be flushed before the hardware uses them.
- Min or max operations flush denorms for comparison, but the result may or may not be denorm flushed.
- NaN input to an operation always produces NaN on output. But the exact bit pattern of the NaN is not required to stay the same (unless the operation is a raw move instruction - which doesn't alter data.)
- Min or max operations for which only one operand is NaN return the other operand as the result (contrary to comparison rules we looked at earlier). This is an IEEE 754R rule.

The arithmetic rules in Direct3D 10 and later don't make any distinctions between "quiet" and "signaling" NaN values (QNaN vs SNaN). All "NaN" values are handled the same way.

- If both inputs to min() or max() are NaN, then any NaN is returned.
- An IEEE 754R rule is that  $\min(-0,+0) == \min(+0,-0) == -0$ , and  $\max(-0,+0) == \max(+0,-0) == +0$ ; which honor the sign. That's in contrast to the comparison rules for signed zero (stated above). Direct3D 11 recommends the IEEE 754R behavior here, but doesn't enforce it; it's permissible for the result of comparing zeros to be dependent on the order of parameters, using a comparison that ignores the signs.

- $x * 1.0f$  always results in  $x$  (except denorm flushed).
- $x / 1.0f$  always results in  $x$  (except denorm flushed).
- $x \pm 0.0f$  always results in  $x$  (except denorm flushed). But  $-0 + 0 = +0$ .
- Fused operations (such as `mad`, `dp3`) produce results that are no less accurate than the worst possible serial ordering of evaluation of the unfused expansion of the operation. The definition of the worst possible ordering, for the purpose of tolerance, is not a fixed definition for a given fused operation; it depends on the particular values of the inputs. The individual steps in the unfused expansion are each allowed 1 ULP tolerance (or for any instructions Direct3D calls out with a more lax tolerance than 1 ULP, the more lax tolerance is allowed).
- Fused operations adhere to the same NaN rules as non-fused operations.
- `sqrt` and `rcp` have 1 ULP tolerance. The shader reciprocal and reciprocal square-root instructions, `rcp` and `rsq`, have their own separate relaxed precision requirement.
- Multiply and divide each operate at the 32-bit floating-point precision level (accuracy to 0.5 ULP for multiply, 1.0 ULP for reciprocal). If  $x/y$  is implemented directly, results must be of greater or equal accuracy than a two-step method.

## 64-bit (double precision) floating point rules

Hardware and display drivers optionally support double-precision floating-point. To indicate support, when you call [ID3D11Device::CheckFeatureSupport](#) with `D3D11_FEATURE_DOUBLES`, the driver sets `DoublePrecisionFloatShaderOps` of `D3D11_FEATURE_DATA_DOUBLES` to TRUE. The driver and hardware must then support all double-precision floating-point instructions.

Double-precision instructions follow IEEE 754R behavior requirements.

Support for generation of denormalized values is required for double-precision data (no flush-to-zero behavior). Likewise, instructions don't read denormalized data as a signed zero, they honor the denorm value.

## 16-bit floating-point rules

Direct3D 11 also supports 16-bit representations of floating-point numbers.

Format:

- 1 sign bit (s) in the MSB bit position
- 5 bits of biased exponent (e)
- 10 bits of fraction (f), with an additional hidden bit

A float16 value (v) follows these rules:

- if  $e == 31$  and  $f != 0$ , then  $v$  is NaN regardless of s
- if  $e == 31$  and  $f == 0$ , then  $v = (-1)s * \infty$  (signed infinity)
- if e is between 0 and 31, then  $v = (-1)s * 2^{(e-15)} * (1.f)$
- if  $e == 0$  and  $f != 0$ , then  $v = (-1)s * 2^{(e-14)} * (0.f)$  (denormalized numbers)
- if  $e == 0$  and  $f == 0$ , then  $v = (-1)s * 0$  (signed zero)

32-bit floating-point rules also hold for 16-bit floating-point numbers, adjusted for the bit layout described earlier. Exceptions to this include:

- Precision: Unfused operations on 16-bit floating-point numbers produce a result that is the nearest representable value to an infinitely-precise result (round to nearest even, per IEEE-754, applied to 16-bit values). 32-bit floating-point rules adhere to 1 ULP tolerance, 16-bit floating-point rules adhere to 0.5 ULP for unfused operations, and 0.6 ULP for fused operations.
- 16-bit floating-point numbers preserve denorms.

## 11-bit and 10-bit floating-point rules

Direct3D 11 also supports 11-bit and 10-bit floating-point formats.

Format:

- No sign bit
- 5 bits of biased exponent (e)
- 6 bits of fraction (f) for an 11-bit format, 5 bits of fraction (f) for a 10-bit format, with an additional hidden bit in either case.

A float11/float10 value (v) follows the following rules:

- if  $e == 31$  and  $f != 0$ , then  $v$  is NaN
- if  $e == 31$  and  $f == 0$ , then  $v = +\infty$
- if e is between 0 and 31, then  $v = 2^{(e-15)} * (1.f)$
- if  $e == 0$  and  $f != 0$ , then  $v = 2^{(e-14)} * (0.f)$  (denormalized numbers)
- if  $e == 0$  and  $f == 0$ , then  $v = 0$  (zero)

32-bit floating-point rules also hold for 11-bit and 10-bit floating-point numbers, adjusted for the bit layout described earlier. Exceptions include:

- Precision: 32-bit floating-point rules adhere to 0.5 ULP.
- 10/11-bit floating-point numbers preserve denorms.
- Any operation that would result in a number less than zero is clamped to zero.

## Related topics

[Resources](#)

[Textures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tiled resources

Article • 08/23/2019

Tiled resources can be thought of as large logical resources that use small amounts of physical memory.

This section describes why tiled resources are needed and how you create and use tiled resources.

## In this section

Topic	Description
<a href="#">Why are tiled resources needed?</a>	Tiled resources are needed so less graphics processing unit (GPU) memory is wasted storing regions of surfaces that the application knows will not be accessed, and the hardware can understand how to filter across adjacent tiles.
<a href="#">Creating tiled resources</a>	Tiled resources are created by specifying the <a href="#">D3D11_RESOURCE_MISC_TILED</a> flag when you create a resource.
<a href="#">Tiled Resource APIs</a>	The APIs described in this section work with tiled resources and tile pool.
<a href="#">Pipeline access to tiled resources</a>	Tiled resources can be used in shader resource views (SRV), render target views (RTV), depth stencil views (DSV) and unordered access views (UAV), as well as some bind points where views aren't used, such as vertex buffer bindings.
<a href="#">Tiled resources features tiers</a>	Direct3D 11.2 exposes tiled resources support in two tiers with the <a href="#">D3D11_TILED_RESOURCES_TIER</a> values.

## Related topics

[Resources](#)

## Feedback



Was this page helpful?  

Get help at Microsoft Q&A

# Why are tiled resources needed?

Article • 08/19/2020

Tiled resources are needed so less graphics processing unit (GPU) memory is wasted storing regions of surfaces that the application knows will not be accessed, and the hardware can understand how to filter across adjacent tiles.

In a graphics system (that is, the operating system, display driver, and graphics hardware) without tiled resource support, the graphics system manages all Direct3D memory allocations at subresource granularity. For a [Buffer](#), the entire Buffer is the subresource. For a [Texture](#) (for example, [Texture2D](#)), each mip level is a subresource; for a texture array (for example, [Texture2DArray](#)), each mip level at a given array slice is a subresource. The graphics system only exposes the ability to manage the mapping of allocations at this subresource granularity. In the context of tiled resources, "mapping" refers to making data visible to the GPU.

Suppose an application knows that a particular rendering operation only needs to access a small portion of an image mipmap chain (perhaps not even the full area of a given mipmap). Ideally, the app could inform the graphics system about this need. The graphics system would then only bother to ensure that the needed memory is mapped on the GPU without paging in too much memory. In reality, without tiled resource support, the graphics system can only be informed about the memory that needs to be mapped on the GPU at subresource granularity (for example, a range of full mipmap levels that could be accessed). There is no demand faulting in the graphics system either, so potentially a lot of excess GPU memory must be used to make full subresources mapped before a rendering command that references any part of the memory is executed. This is just one issue that makes the use of large memory allocations difficult in Direct3D without tiled resource support.

Direct3D 11 supports [Texture2D](#) surfaces with up to 16384 pixels on a given side. An image that is 16384 wide by 16384 tall and 4 bytes per pixel would consume 1GB of video memory (and adding mipmaps would double that amount). In practice, all 1GB would rarely need to be referenced in a single rendering operation.

Some game developers model terrain surfaces as large as 128K by 128K. The way they get this to work on existing GPUs is to break the surface into tiles that are small enough for hardware to handle. The application must figure out which tiles might be needed and load them into a cache of textures on the GPU - a software paging system. A significant downside to this approach comes from the hardware not knowing anything about the paging that is going on: When a part of an image needs to be shown on screen that straddles tiles, the hardware does not know how to perform fixed function

(that is, efficient) filtering across tiles. This means the application managing its own software tiling must resort to manual texture filtering in shader code (which becomes very expensive if a good quality anisotropic filter is desired) and/or waste memory authoring gutters around tiles that contain data from neighboring tiles so that fixed function hardware filtering can continue to provide some assistance.

If a tiled representation of surface allocations could be a first class feature in the graphics system, the application could tell the hardware which tiles to make available. In this way, less GPU memory is wasted storing regions of surfaces that the application knows will not be accessed, and the hardware can understand how to filter across adjacent tiles, alleviating some of the pain experienced by developers who perform software tiling on their own.

But to provide a complete solution, something must be done to deal with the fact that, independent of whether tiling within a surface is supported, the maximum surface dimension is currently 16384 - nowhere near the 128K+ that applications already want. Just requiring the hardware to support larger texture sizes is one approach, however there are significant costs and/or tradeoffs to going this route. Direct3D 11's texture filter path and rendering path are already saturated in terms of precision in supporting 16K textures with the other requirements, such as supporting viewport extents falling off the surface during rendering, or supporting texture wrapping off the surface edge during filtering. A possibility is to define a tradeoff such that as the texture size increases beyond 16K, functionality/precision is given up in some manner. Even with this concession however, additional hardware costs might be required in terms of addressing capability throughout the hardware system to go to larger texture sizes.

One issue that comes into play as textures get very large is that single precision floating point texture coordinates (and the associated interpolators to support rasterization) run out of precision to specify locations on the surface accurately. Jittery texture filtering would ensue. One expensive option would be to require double precision interpolator support, though that could be overkill given a reasonable alternative.

An alternate name for tiled resources is "sparse texture." "Sparse" conveys both the tiled nature of the resources as well as perhaps the primary reason for tiling them - that not all of them are expected to be mapped at once. In fact, an application could conceivably author a tiled resource in which no data is authored for all regions+mips of the resource, intentionally. So, the content itself could be sparse, and the mapping of the content in GPU memory at a given time would be a subset of that (even more sparse).

Another scenario that could be served by tiled resources is enabling multiple resources of different dimensions/formats to share the same memory. Sometimes applications have exclusive sets of resources that are known not to be used at the same time, or

resources that are created only for very brief use and then destroyed, followed by creation of other resources. A form of generality that can fall out of "tiled resources" is that it is possible to allow the user to point multiple different resources at the same (overlapping) memory. In other words, the creation and destruction of "resources" (which define a dimension/format and so on) can be decoupled from the management of the memory underlying the resources from the application's point of view.

## Related topics

[Tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Creating tiled resources

Article • 11/13/2019

Tiled resources are created by specifying the [D3D11\\_RESOURCE\\_MISC\\_TILED](#) flag when you create a resource.

Restrictions on when you can use [D3D11\\_RESOURCE\\_MISC\\_TILED](#) to create a resource are described in [Tiled resource creation parameters](#).

A non-tiled resource's storage is allocated in the graphics system when the resource is created. For example, when you call [ID3D11Device::CreateTexture2D](#) to create an array of 2D textures, the graphics system allocates storage for those 2D textures. When a tiled resource is created, the graphics system doesn't allocate the storage for the resource contents. Instead, when an application creates a tiled resource, the graphics system makes an address space reservation for the tiled surface's area only, and then allows the mapping of the tiles to be controlled by the application. The "mapping" of a tile is simply the physical location in memory that a logical tile in a resource points to (or **NULL** for an unmapped tile). Don't confuse this concept with the notion of mapping a Direct3D resource for CPU access, which despite using the same name is completely independent. You will be able to define and change the mapping of each tile individually as needed, knowing that all tiles for a surface don't need to be mapped at a time, thereby making effective use of the amount of memory available.

This section provides more info about how to create tiled resources.

## In this section

[+] [Expand table](#)

Topic	Description
<a href="#">Mappings are into a tile pool</a>	When a resource is created with the <a href="#">D3D11_RESOURCE_MISC_TILED</a> flag, the tiles that make up the resource come from pointing at locations in a tile pool. A tile pool is a pool of memory (backed by one or more allocations behind the scenes - unseen by the application).
<a href="#">Tiled resource creation parameters</a>	There are some constraints on the type of Direct3D resources that you can create with the <a href="#">D3D11_RESOURCE_MISC_TILED</a> flag. This section provides the valid parameters for creating tiled resources.
<a href="#">Address space available for tiled resources</a>	This section specifies the virtual address space that is available for tiled resources.

Topic	Description
Tile pool creation parameters	Use the parameters in this section to define tile pools via the <a href="#">ID3D11Device::CreateBuffer</a> API.
Tiled resource cross process and device sharing	Tile pools can be shared with other processes just like traditional resources. Tiled resources that reference tile pools can't be shared across devices and processes. But separate processes can create their own tiled resources that map to tile pools that are shared between those tiled resources.
Operations available on tiled resources	This section lists operations that you can perform on tiled resources.
Operations available on tile pools	This section lists operations that you can perform on tile pools.
How a tiled resource's area is tiled	When you create a tiled resource, the dimensions, format element size, and number of mipmaps and/or array slices (if applicable) determine the number of tiles that are required to back the entire surface area.

## Related topics

[Tiled resources](#)

---

## Feedback

Was this page helpful?



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

# Mappings are into a tile pool

Article • 08/19/2020

When a resource is created with the [D3D11\\_RESOURCE\\_MISC\\_TILED](#) flag, the tiles that make up the resource come from pointing at locations in a tile pool. A tile pool is a pool of memory (backed by one or more allocations behind the scenes - unseen by the application). The operating system and display driver manage this pool of memory, and the memory footprint is easily understood by an application. Tiled resources map 64KB regions by pointing to locations in a tile pool. One fallout of this setup is it allows multiple resources to share and reuse the same tiles, and also for the same tiles to be reused at different locations within a resource if desired.

The cost for the flexibility of populating the tiles for a resource out of a tile pool is that the resource has to do the work of defining and maintaining the mapping of which tiles in the tile pool represent the tiles needed for the resource. Tile mappings can be changed. Also, not all tiles in a resource need to be mapped at a time; a resource can have **NULL** mappings. A **NULL** mapping defines a tile as not being available from the point of view of the resource accessing it.

Multiple tile pools can be created, and any number of tiled resources can map into any given tile pool at the same time. Tile pools can also be grown or shrunk. For more info, see [Tile pool resizing](#). One constraint that exists to simplify display driver and runtime implementation is that a given tiled resource can only have mappings into at most one tile pool at a time (as opposed to having simultaneous mapping to multiple tile pools).

The amount of storage that is associated with a tiled resource itself (that is, independent tile-pool memory) is roughly proportional to the number of tiles actually mapped to the pool at any given time. In hardware, this fact boils down to scaling the memory footprint for page table storage roughly with the amount of tiles that are mapped (for example, using a multilevel page table scheme as appropriate).

The tile pool can be thought of as an entirely software abstraction that enables Direct3D applications to effectively be able to program the page tables on the graphics processing unit (GPU) without having to know the low level implementation details (or deal with pointer addresses directly). Tile pools don't apply any additional levels of indirection in hardware. Optimizations of a single level page table using constructs like page directories are independent of the tile pool concept.

Let us explore what storage the page table itself could require in the worst case (though in practice implementations only require storage roughly proportional to what is mapped).

Suppose each page table entry is 64 bits.

For the worst-case page table size hit for a single surface, given the resource limits in Direct3D 11, suppose a tiled resource is created with a 128 bit-per-element format (for example, a RGBA float), so a 64KB tile contains only 4096 pixels. The maximum supported [Texture2DArray](#) size of 16384\*16384\*2048 (but with only a single mipmap) would require about 1GB of storage in the page table if fully populated (not including mipmaps) using 64 bit table entries. Adding mipmaps would grow the fully-mapped (worst case) page table storage by about a third, to about 1.3GB.

This case would give access to about 10.6 terabytes of addressable memory. There might be a limit on the amount of addressable memory however, which would reduce these amounts, perhaps to around the terabyte range.

Another case to consider is a single [Texture2D](#) tiled resource of 16384\*16384 with a 32 bit-per-element format, including mipmaps. The space needed in a fully populated page table would be roughly 170KB with 64 bit table entries.

Finally, consider an example using a BC format, say BC7 with 128 bits per tile of 4x4 pixels. That is one byte per pixel. A [Texture2DArray](#) of 16384\*16384\*2048 including mipmaps would require roughly 85MB to fully populate this memory in a page table. That is not bad considering this allows one tiled resource to span 550 gigapixels (512 GB of memory in this case).

In practice, nowhere near these full mappings would be defined given that the amount of physical memory available wouldn't allow anywhere near that much to be mapped and referenced at a time. But with a tile pool, applications could choose to reuse tiles (as a simple example, reusing a "black" colored tile for large black regions in an image) - effectively using the tile pool (that is, page table mappings) as a tool for memory compression.

The initial contents of the page table are **NULL** for all entries. Applications also can't pass initial data for the memory contents of the surface since it starts off with no memory backing.

## In this section

Topic	Description
Tile pool creation	A tile pool is created via the <a href="#">ID3D11Device::CreateBuffer</a> API by passing the <a href="#">D3D11_RESOURCE_MISC_TILE_POOL</a> flag in the <b>MiscFlags</b> member of the <a href="#">D3D11_BUFFER_DESC</a> structure that the <i>pDesc</i> parameter points to.

Topic	Description
Tile pool resizing	Use the <a href="#">ID3D11DeviceContext2::ResizeTilePool</a> API to grow a tile pool if the application needs more working set for the tiled resources mapping into it or to shrink if less space is needed.
Hazard tracking versus tile pool resources	For non-tiled resources, Direct3D can prevent certain hazard conditions during rendering, but because hazard tracking would be at a tile level for tiled resources, tracking hazard conditions during rendering of tiled resources might be too expensive.

## Related topics

[Creating tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tile pool creation

Article • 08/23/2019

A tile pool is created via the [ID3D11Device::CreateBuffer](#) API by passing the [D3D11\\_RESOURCE\\_MISC\\_TILE\\_POOL](#) flag in the [MiscFlags](#) member of the [D3D11\\_BUFFER\\_DESC](#) structure that the *pDesc* parameter points to.

Applications can create one or more tile pools per Direct3D device. The total size of each tile pool is restricted to Direct3D 11's resource size limit, which is roughly 1/4 of graphics processing unit (GPU) RAM.

A tile pool is made of 64KB tiles, but the operating system (display driver) manages the entire pool as one or more allocations behind the scenes—the breakdown is not visible to applications. Tiled resources define content by pointing at tiles within a tile pool. Unmapping a tile from a tiled resource is done by pointing the tile to **NULL**. Such unmapped tiles have rules about the behavior of reads or writes. For info, see [Hazard tracking versus tile pool resources](#).

## Related topics

[Mappings are into a tile pool](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tile pool resizing

Article • 08/23/2019

Use the [ID3D11DeviceContext2::ResizeTilePool](#) API to grow a tile pool if the application needs more working set for the tiled resources mapping into it or to shrink if less space is needed. Another option for applications is to allocate additional tile pools for new tiled resources. But if any single tiled resource needs more space than initially available in its tile pool, growing the tile pool is a good option. A tiled resource can't have mappings into multiple tile pools at the same time.

When a tile pool is grown, additional tiles are added to the end via one or more new allocations by the display driver. This breakdown into allocations isn't visible to the application. Existing memory in the tile pool is left untouched, and existing tiled resource mappings into that memory remain intact.

When a tile pool is shrunk, tiles are removed from the end. Tiles are removed even below the initial allocation size, down to 0, which means new mappings can't be made past the new size. But, existing mappings past the end of the new size remain intact and useable. The display driver will keep the memory around as long as mappings to any part of the allocations that the driver uses for the tile pool memory remains. If after shrinking some memory has been kept alive because tile mappings are pointing to it and then the tile pool is regrown again (by any amount), the existing memory is reused first before any additional allocations occur to service the size of the grow operation.

To be able to save memory, an application has to not only shrink a tile pool but also remove/remap existing mappings past the end of the new smaller tile pool size.

The act of shrinking (and removing mappings) doesn't necessarily produce immediate memory savings. Freeing of memory depends on how granular the display driver's underlying allocations for the tile pool are. When shrinking happens to be enough to make a display driver allocation unused, the display driver can free it. If a tile pool was grown, shrinking to previous sizes (and removing/remapping tile mappings correspondingly) is most likely to yield memory savings, though not guaranteed in the case that the sizes don't exactly align with the underlying allocation sizes chosen by the display driver.

## Related topics

[Mappings are into a tile pool](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Hazard tracking versus tile pool resources

Article • 08/23/2019

For non-tiled resources, Direct3D can prevent certain hazard conditions during rendering, but because hazard tracking would be at a tile level for tiled resources, tracking hazard conditions during rendering of tiled resources might be too expensive.

For example, for non-tiled resources, the runtime doesn't allow any given SubResource to be bound as an input (such as a ShaderResourceView) and as an output (such as a RenderTargetView) at the same time. If such a case is encountered, the runtime unbinds the input. This tracking overhead in the runtime is cheap and is done at the SubResource level. One of the benefits of this tracking overhead is to minimize the chances of applications accidentally depending on hardware shader execution order. Hardware shader execution order can vary if not on a given graphics processing unit (GPU), then certainly across different GPUs.

Tracking how resources are bound might be too expensive for tiled resources because tracking is at a tile level. New issues arise such as possibly validating away attempts to render to an RenderTargetView with one tile mapped to multiple areas in the surface simultaneously. If it turns out this per-tile hazard tracking is too expensive for the runtime, ideally this would at least be an option in the debug layer.

An application must inform the display driver when it has issued a write or read operation to a tiled resource that references tile pool memory that will also be referenced by separate tiled resources in upcoming read or write operations that it is expecting the first operation to complete before the following operations can begin. For more info about this condition, see [ID3D11DeviceContext2::TiledResourceBarrier](#) remarks.

## Related topics

[Mappings are into a tile pool](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Tiled resource creation parameters

Article • 11/13/2019

There are some constraints on the type of Direct3D resources that you can create with the [D3D11\\_RESOURCE\\_MISC\\_TILED](#) flag. This section provides the valid parameters for creating tiled resources.

## Supported Resource Type

Texture2D[Array] (including TextureCube[Array], which is a variant of Texture2D[Array]) or Buffer.

**NOT supported:** Texture1D[Array] or Texture3D, but Texture3D might be supported in the future.

## Supported Resource Usage

D3D11\_USAGE\_DEFAULT.

**NOT supported:** D3D11\_USAGE\_DYNAMIC, D3D11\_USAGE\_STAGING, or D3D11\_USAGE\_IMMUTABLE.

## Supported Resource Misc Flags

D3D11\_RESOURCE\_MISC\_TILED (by definition), \_MISC\_TEXTURECUBE, \_DRAWINDIRECT\_ARGS, \_BUFFER\_ALLOW\_RAW\_VIEWS, \_BUFFER\_STRUCTURED, \_RESOURCE\_CLAMP, or \_GENERATE\_MIPS.

**NOT supported:** \_SHARED, \_SHARED\_KEYEDMUTEX, \_GDI\_COMPATIBLE, \_SHARED\_NTHANDLE, \_RESTRICTED\_CONTENT, \_RESTRICT\_SHARED\_RESOURCE, \_RESTRICT\_SHARED\_RESOURCE\_DRIVER, \_GUARDED, or \_TILE\_POOL.

## Supported Bind Flags

D3D11\_BIND\_SHADER\_RESOURCE, \_RENDER\_TARGET, \_DEPTH\_STENCIL, or \_UNORDERED\_ACCESS.

**NOT supported:** \_CONSTANT\_BUFFER, \_VERTEX\_BUFFER [note that binding a tiled Buffer as an SRV/UAV/RTV is still ok], \_INDEX\_BUFFER, \_STREAM\_OUTPUT, \_BIND\_DECODER, or \_BIND\_VIDEO\_ENCODER.

## Supported Formats

All formats that would be available for the given configuration regardless of it being tiled, with some exceptions.

## Supported SampleDesc (Multisample count, quality)

Whatever would be supported for the given configuration regardless of it being tiled, with some exceptions.

## Supported Width/Height/MipLevels/ArraySize

Full extents supported by Direct3D 11. Tiled resources don't have the restriction on total memory size imposed on non-tiled resources. Tiled resources are only constrained by overall virtual address space limits. For info, see [Address space available for tiled resources](#).

The initial contents of tile pool memory are undefined.

## Related topics

[Creating tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Address space available for tiled resources

Article • 11/13/2019

This section specifies the virtual address space that is available for tiled resources.

On 64-bit operating systems, at least 40 bits of virtual address space (1 Terabyte) is available.

For 32-bit operating systems, the address space is 32 bit (4 GB). For 32-bit ARM systems, individual tiled resource creation can fail if the allocation would use more than 27 bits of address space (128 MB). This includes any hidden padding in the address space the hardware may use for mipmaps, packed tile padding, and possibly padding surface dimensions to powers of 2.

On graphics systems with a separate page table for the graphics processing unit (GPU), most of this address space will be available to GPU resources made by the application, though GPU allocations made by the display driver fit in the same space.

On future systems with a page table shared between the CPU and GPU, the available address space is shared between all CPU and GPU allocations in a process.

## Related topics

[Tiled resource creation parameters](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tile pool creation parameters

Article • 08/23/2019

Use the parameters in this section to define tile pools via the [ID3D11Device::CreateBuffer](#) API.

## Size

Allocation size, as a multiple of 64KB. 0 is valid because you can later use a [ID3D11DeviceContext2::ResizeTilePool](#) operation.

## Supported Resource Misc Flags

`D3D11_RESOURCE_MISC_TILE_POOL` (identifies the resource as a tile pool), `D3D11_RESOURCE_MISC_SHARED`, `_SHARED_KEYEDMUTEX`, or `_SHARED_NTHANDLE`.

## Supported Resource Usage

`D3D11_USAGE_DEFAULT` only.

## Related topics

[Creating tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tiled resource cross process and device sharing

Article • 08/23/2019

Tile pools can be shared with other processes just like traditional resources. Tiled resources that reference tile pools can't be shared across devices and processes. But separate processes can create their own tiled resources that map to tile pools that are shared between those tiled resources.

Shared tile pools can't be resized.

## In this section

Topic	Description
<a href="#">Stencil formats not supported with tiled resources</a>	Formats that contain stencil aren't supported with tiled resources.

## Related topics

[Creating tiled resources](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# Stencil formats not supported with tiled resources

Article • 08/23/2019

Formats that contain stencil aren't supported with tiled resources.

Formats that contain stencil include DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT (and related formats in the R24G8 family) and DXGI\_FORMAT\_D32\_FLOAT\_S8X24\_UINT (and related formats in the R32G8X24 family).

Some implementations store depth and stencil in separate allocations while others store them together. Tile management for the two schemes would have to be different, and no single API can abstract or rationalize the differences. We recommend for future hardware to support independent depth and stencil surfaces, each independently tiled. 32 bit depth would have 128x128 tiles, and 8 bit stencil would have 256x256 tiles. Therefore, applications would have to live with tile shape misalignment between depth and stencil. But the same problem exists with different render target surface formats already.

## Related topics

[Tiled resource cross process and device sharing](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Operations available on tiled resources

Article • 08/23/2019

This section lists operations that you can perform on tiled resources.

- void [ID3D11DeviceContext2::UpdateTileMappings](#) and [ID3D11DeviceContext2::CopyTileMappings](#) operations - These operations point tile locations in a tiled resource to locations in tile pools, or to NULL, or to both. These operations can update a disjoint subset of the tile pointers.
- Copy\*() and Update\*() operations - All the APIs that can copy data to and from a default pool surface (for example, [ID3D11DeviceContext1::CopySubresourceRegion1](#) and [ID3D11DeviceContext1::UpdateSubresource1](#)) work for tiled resources. Reading from unmapped tiles produces 0 and writes to unmapped tiles are dropped.
- [ID3D11DeviceContext2::CopyTiles](#) and [ID3D11DeviceContext2::UpdateTiles](#) operations - These operations exist for copying tiles at 64KB granularity to and from any tiled resource and a buffer resource in a canonical memory layout. The display driver and hardware perform any memory "swizzling" necessary for the tiled resource.
- Direct3D pipeline bindings and view creations / bindings that would work on non-tiled resources work on tiled resources as well.

Tile controls are available on immediate or deferred contexts (just like updates to typical resources) and upon execution impact subsequent accesses to the tiles (not previously submitted operations).

## Related topics

[Creating tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Operations available on tile pools

Article • 08/19/2020

This section lists operations that you can perform on tile pools.

- The lifetime of tile pools works like any other Direct3D Resource, backed by reference counting, including in this case tracking of mappings from tiled resources. When the application no longer references a tile pool and any tile mappings to the memory are gone and graphics processing unit (GPU) accesses completed, the operating system will deallocate the tile pool.
- APIs related to surface sharing and synchronization work for tile pools (but not directly on tiled resources). Similar to the behavior for offered tile pools, Direct3D commands that access tiled resources that point to a tile pool are dropped if the tile pool has been shared and is currently acquired by another device and process.
- [ID3D11DeviceContext2::ResizeTilePool](#) operation
- [IDXGIDevice2::OfferResources](#) and [ReclaimResources](#) operations - These APIs for yielding memory temporarily to the system operate on the entire tile pool (and aren't available for individual tiled resources). If a tiled resource points to a tile in an offered tile pool, the tiled resource behaves as if it is offered (for example, the runtime drops commands that reference it).

Data can't be copied to and from tile pool memory directly. Accesses to the memory are always done through tiled resources.

## Related topics

[Creating tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How a tiled resource's area is tiled

Article • 08/19/2020

When you create a tiled resource, the dimensions, format element size, and number of mipmaps and/or array slices (if applicable) determine the number of tiles that are required to back the entire surface area. The pixel/byte layout within tiles is determined by the implementation. The number of pixels that fit in a tile, depending on the format element size, is fixed and identical whether you use a standard swizzle or not.

The number of tiles that will be used by a given surface size and format element width is well defined and predictable based on the tables in the following sections. For resources that contain mipmaps or cases where surface dimensions don't fill a tile, some constraints exist and are discussed in [Mipmap packing](#).

Different tiled resources can point to identical memory with different formats as long as applications don't rely on the results of writing to the memory with one format and reading with another. But, in constrained circumstances, applications can rely on the results of writing to the memory with one format and reading with another if the formats are in the same format family (that is, they have the same typeless parent format). For example, DXGI\_FORMAT\_R8G8B8A8\_UNORM and DXGI\_FORMAT\_R8G8B8A8\_UINT are compatible with each other but not with DXGI\_FORMAT\_R16G16\_UNORM. An application must conservatively match all resource properties in order to reinterpret data between two textures since tile patterns are very conservatively undefined. Obviously, though, the format is more lax. The formats need only be compatible as illustrated above. An exception is where bleeding data from one format aliasing to another is well defined: if a tile completely contains 0 for all its bits, that tile can be used with any format that interprets those memory contents as 0 (regardless of memory layout). So, a tile could be cleared to 0x00 with the format DXGI\_FORMAT\_R8\_UNORM and then used with a format like DXGI\_FORMAT\_R32G32\_FLOAT and it would appear the contents are still (0.0f,0.0f).

The layout of data within a tile may be depend on resource properties, the subresource in which it resides, and the tile location within the subresource. The major exceptions are illustrated above: compatible formats with other resource properties being equal and when all texels are the same pattern.

## In this section

Topic	Description

Topic	Description
Texture2D and Texture2DArray subresource tiling	These tables show how <a href="#">Texture2D</a> and <a href="#">Texture2DArray</a> subresources are tiled.
Texture3D subresource tiling	This table shows how <a href="#">Texture3D</a> subresources are tiled.
Buffer tiling	A <a href="#">Buffer</a> resource is divided into 64KB tiles, with some empty space in the last tile if the size is not a multiple of 64KB.
Mipmap packing	Depending on the <a href="#">tier</a> of tiled resources support, mipmaps with certain dimensions don't follow the standard tile shapes and are considered to all be packed together with one another in a manner that is opaque to the application.

## Related topics

[Creating tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Texture2D and Texture2DArray subresource tiling

Article • 08/19/2020

These tables show how [Texture2D](#) and [Texture2DArray](#) subresources are tiled. The values in these tables don't count tail mip packing.

This table shows how [Texture2D](#) and [Texture2DArray](#) subresources with multisample counts of 1 are tiled.

Bits/Pixel (1 sample/pixel)	Tile Dimensions (Pixels, WxH)
8	256x256
16	256x128
32	128x128
64	128x64
128	64x64
BC1,4	512x256
BC2,3,5,6,7	256x256

Format bit counts not supported with tiled resources are 96 bpp formats, video formats, DXGI\_FORMAT\_R1\_UNORM, DXGI\_FORMAT\_R8G8\_B8G8\_UNORM, and DXGI\_FORMAT\_R8R8\_G8B8\_UNORM.

This table shows how [Texture2D](#) and [Texture2DArray](#) subresources with various multisample counts are tiled.

Multisample Count	Divide Tile Dimensions Above by (WxH)
1	1x1
2	2x1
4	2x2
8	4x2
16	4x4

Only sample counts 1 and 4 are required (and allowed) to be supported with tiled resources. Tiled resources don't currently support 2, 8, and 16 even though they are shown.

Implementations can choose to support 2, 8, and/or 16 sample multisample antialiasing (MSAA) mode for non-tiled resources even though tiled resource don't support them.

Tiled resources with sample counts larger than 1 can't use 128 bpp formats.

## Related topics

[How a tiled resource's area is tiled](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Texture3D subresource tiling

Article • 08/19/2020

This table shows how [Texture3D](#) subresources are tiled. The values in this table don't count tail mip packing.

This table takes the [Texture2D](#) tiling and divides the x/y dimensions by 4 each and adds 16 layers of depth. All the tiles for the first plane (2D plane of tiles defining the first 16 layers of depth) appear before the subsequent planes.

## ⓘ Note

Texture3D support in tiled resources isn't exposed in the initial implementation of tiled resources, but the desired tile shapes are listed here because support might be consideration in a future release.

Bits/Pixel (1 sample/pixel)	Tile Dimensions (Pixels, WxHxD)
8	64x32x32
16	32x32x32
32	32x32x16
64	32x16x16
128	16x16x16
BC1,4	128x64x16
BC2,3,5,6,7	64x64x16

Format bit counts not supported with tiled resources are 96 bpp formats, video formats, DXGI\_FORMAT\_R1\_UNORM, DXGI\_FORMAT\_R8G8\_B8G8\_UNORM, and DXGI\_FORMAT\_R8R8\_G8B8\_UNORM.

## Related topics

[How a tiled resource's area is tiled](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Buffer tiling

Article • 08/19/2020

A [Buffer](#) resource is divided into 64KB tiles, with some empty space in the last tile if the size is not a multiple of 64KB.

Structured buffers must have no constraint on the stride to be tiled. But possible performance optimizations in hardware for using [StructuredBuffers](#) can be sacrificed by making them tiled in the first place.

## Related topics

[How a tiled resource's area is tiled](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Mipmap packing

Article • 08/23/2019

Depending on the [tier](#) of tiled resources support, mipmaps with certain dimensions don't follow the standard tile shapes and are considered to all be packed together with one another in a manner that is opaque to the application. Higher tiers of support have broader guarantees about what types of surface dimensions fit in the standard tile shapes (and can therefore be individually mapped by applications).

What can vary between implementations is that—given a tiled resource's dimensions, format, number of mipmaps, and array slices—some number M of mips (per array slice) can be packed into some number N tiles. The [ID3D11Device2::GetResourceTiling](#) API exists to allow the driver to report to the application what M and N are (among other details about the surface that this API reports that are standard and don't vary by hardware vendor). The set of tiles for the packed mips are still 64KB and can be individually mapped into disparate locations in a tile pool. But the pixel shape of the tiles and how the mipmaps fit across the set of tiles is specific to a hardware vendor and too complex to expose. So, applications are required to either map all of the tiles that are designated as packed, or none of them, at a time. Otherwise, the behavior for accessing the tiled resource is undefined.

For arrayed surfaces, the set of packed mips and the number of packed tiles storing those mips (M and N described preceding) applies individually for each array slice.

Dedicated APIs for copying tiles ([ID3D11DeviceContext2::CopyTiles](#) and [ID3D11DeviceContext2::UpdateTiles](#)) can't access packed mips. Applications that want to copy data to and from packed mips can do so using all the non-tiled resource specific APIs for copying and rendering to surfaces.

## Related topics

[How a tiled resource's area is tiled](#)

---

## Feedback



Was this page helpful?

Get help at Microsoft Q&A

# Tiled Resource APIs

Article • 08/23/2019

The APIs described in this section work with tiled resources and tile pool.

- [Assigning tiles from a tile pool to a resource](#)
- [Querying resource tiling and support](#)
- [Copying tiled data](#)
- [Resizing tile pool](#)
- [Tiled resource barrier](#)
- [Related topics](#)

## Assigning tiles from a tile pool to a resource

The [ID3D11DeviceContext2::UpdateTileMappings](#) and [ID3D11DeviceContext2::CopyTileMappings](#) APIs manipulate and query tile mappings. Update calls only affect the tiles identified in the call, and others are left as defined previously.

Any given tile from a tile pool can be mapped to multiple locations in a resource and even multiple resources. This mapping includes tiles in a resource that have an implementation-chosen layout ([Mipmap packing](#)) where multiple mipmaps are packed together into a single tile. The catch is that if data is written to the tile via one mapping, but read via a differently configured mapping, the results are undefined. Careful use of this flexibility can still be useful for an application though, like sharing a tile between resources that will not be used simultaneously, where the contents of the tile are always initialized through the same resource mapping as they will be subsequently read from. Similarly, a tile mapped to hold the packed mipmaps of multiple different resources with the same surface dimensions will work fine - the data will appear the same in both mappings.

Changes to tile assignments for a resource can be made at any time in an immediate or deferred context.

## Querying resource tiling and support

To query resource tiling, use [ID3D11Device2::GetResourceTiling](#).

For other resource tiling support, use [ID3D11Device2::CheckMultisampleQualityLevels1](#).

# Copying tiled data

Any methods in Direct3D for moving data around work with tiled resources just as if they are not tiled, except that writes to unmapped areas are dropped and reads from unmapped areas produce 0. 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 writes to multi-mapped tiles are non-deterministic and non-repeatable. That is, accesses happen in whatever order the hardware happens to execute the copy.

Direct3D 11.2 introduces methods for these additional ways to copy:

- Copy between tiles in a tiled resource (at 64KB tile granularity) and (to/from) a buffer in graphics processing unit (GPU) memory (or staging resource) - [ID3D11DeviceContext2::CopyTiles](#)
- Copy from application-provided memory to tiles in a tiled resource - [ID3D11DeviceContext2::UpdateTiles](#)

These methods swizzle/deswizzle as needed, and allow a D3D11\_TILE\_COPY\_NO\_OVERWRITE flag when the caller promises the destination memory is not referenced by GPU work that is in flight.

The tiles involved in the copy can't include tiles that contain packed mipmaps or that have results that are undefined. To transfer data to/from mipmaps that the hardware packs into one tile, you must use the standard (non-tile specific) Copy/Update APIs or [ID3D11DeviceContext::GenerateMips](#) for the whole mip chain.

**Note on [GenerateMips](#):** Using [ID3D11DeviceContext::GenerateMips](#) on a resource with partially mapped tiles will produce results that simply follow the rules for reading and writing **NULL** applied to whatever algorithm the hardware and display driver happen to use to [GenerateMips](#). So, it is not particularly useful for an application to bother doing this unless somehow the areas with **NULL** mappings (and their effect on other mips during the generation phase) will have no consequence on the parts of the surface the application does care about.

Copying tile data from a staging surface or from application memory would be the way to upload tiles that may have been streamed off disk, for example. A variation when streaming off disk is uploading some sort of compressed data to GPU memory and then decoding on the GPU. The decode target could be a buffer resource in GPU memory, from which [CopyTiles](#) then copies to the actual tiled resource. This copy step allows the GPU to swizzle when the swizzle pattern is not known. Swizzling is not needed if the tiled resource itself is a buffer resource (for example, as opposed to a Texture).

The memory layout of the tiles in the non-tiled buffer resource side of the copy is simply linear in memory within 64KB tiles, which the hardware and display driver would swizzle/deswizzle per tile as appropriate when transferring to/from a tiled resource. For multisample antialiasing (MSAA) surfaces, each pixel's samples are traversed in sample-index order before moving 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/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. For specification simplicity, mipmaps smaller than a tile are not packed together in the linear layout. This seems to be a waste of memory space, but as mentioned copying to mips that the hardware packs together is not allowed via [CopyTiles](#) or [UpdateTiles](#). The application can just use generic `UpdateSubresource*`() or `CopySubresource*`() APIs to copy small mips individually, though in the case of `CopySubresource*`() that means the linear memory has to be the same dimension as the tiled resource - `CopySubresource*`() can't copy from a buffer resource to a Texture2D for instance.

If a hardware standard swizzle is defined, flags could be added to indicate that the data in the buffer is to be interpreted in that format (no swizzle necessary on transfer), though alternative approaches to uploading data may also make sense in that case such as allowing applications direct access to tile pool memory.

Copying operations can be done on an immediate or deferred context.

## Resizing tile pool

To resize a tile pool, use [ID3D11DeviceContext2::ResizeTilePool](#).

## Tiled resource barrier

To specify a data access ordering constraint between multiple tiled resources, use [ID3D11DeviceContext2::TiledResourceBarrier](#).

## Related topics

[Tiled resources](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Pipeline access to tiled resources

Article • 08/19/2020

Tiled resources can be used in shader resource views (SRV), render target views (RTV), depth stencil views (DSV) and unordered access views (UAV), as well as some bind points where views aren't used, such as vertex buffer bindings. For the list of supported bindings, see [Tiled resource creation parameters](#). Copy\* operations also work on tiled resources.

If multiple tile coordinates in one or more views is bound to the same memory location, reads and writes from different paths to the same memory will occur in a non-deterministic and non-repeatable order of memory accesses.

If all tiles behind a memory access footprint from a shader are mapped to unique tiles, behavior is identical on all implementations to the surface having the same memory contents in a non-tiled fashion.

This section provides more info about pipeline access to tiled resources.

## In this section

Topic	Description
<a href="#">SRV behavior with non-mapped tiles</a>	Behavior of shader resource view (SRV) reads that involve non-mapped tiles depends on the level of hardware support.
<a href="#">UAV behavior with non-mapped tiles</a>	Behavior of unordered access view (UAV) reads and writes depends on the level of hardware support.
<a href="#">Rasterizer behavior with non-mapped tiles</a>	This section describes rasterizer behavior with non-mapped tiles.
<a href="#">Tile access limitations with duplicate mappings</a>	This section describes tile access limitations with duplicate mappings.
<a href="#">Tiled resources texture sampling features</a>	This section describes tiled resources texture sampling features.
<a href="#">HLSL tiled resources exposure</a>	New Microsoft High Level Shader Language (HLSL) syntax is required to support tiled resources in <a href="#">Shader Model 5</a> .

## Related topics

[Tiled resources](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# SRV behavior with non-mapped tiles

Article • 08/23/2019

Behavior of shader resource view (SRV) reads that involve non-mapped tiles depends on the level of hardware support. For a breakdown of requirements, see read behavior for [Tiled resources features tiers](#). This section summarizes the ideal behavior, which [Tier 2](#) requires.

Consider a texture filter operation that reads from a set of texels in an SRV. Texels that fall on non-mapped tiles contribute 0 in all non-missing components of the format (and the default for missing components) into the overall filter operation alongside contributions from mapped texels. The texels are all weighted and combined together independent of whether the data came from mapped or non-mapped tiles.

Some first generation [Tier 2](#) level hardware does not meet this spec requirement and returns the 0 with defaults described preceding as the overall filter result if any texels (with nonzero weight) fall on non-mapped tiles. No other hardware will be allowed to miss the requirement to include all (nonzero weight) texels in the filter.

## Related topics

[Pipeline access to tiled resources](#)

---

## Feedback

Was this page helpful?



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

# UAV behavior with non-mapped tiles

Article • 08/23/2019

Behavior of unordered access view (UAV) reads and writes depends on the level of hardware support. For a breakdown of requirements, see overall read and write behavior for [Tiled resources features tiers](#). This section summarizes the ideal behavior.

Shader operations that read from a non-mapped tile in a UAV return 0 in all non-missing components of the format, and the default for missing components.

Shader operations that attempt to write to a non-mapped tile cause nothing to be written to the non-mapped area (while writes to a mapped area proceed). This ideal definition for write handling is not required by [Tier 2](#); writes to non-mapped tiles can end up in a cache that subsequent reads could pick up.

## Related topics

[Pipeline access to tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Rasterizer behavior with non-mapped tiles

Article • 08/23/2019

This section describes rasterizer behavior with non-mapped tiles.

## DepthStencilView

Behavior of depth stencil view (DSV) reads and writes depends on the level of hardware support. For a breakdown of requirements, see overall read and write behavior for [Tiled resources features tiers](#).

Here is the ideal behavior:

If a tile isn't mapped in the DepthStencilView, the return value from reading depth is 0, which is then fed into whatever operations are configured for the depth read value. Writes to the missing depth tile are dropped. This ideal definition for write handling is not required by [Tier 2](#); writes to non-mapped tiles can end up in a cache that subsequent reads could pick up.

## RenderTargetView

Behavior of render target view (RTV) reads and writes depends on the level of hardware support. For a breakdown of requirements, see overall read and write behavior for [Tiled resources features tiers](#).

On all implementations, different RTVs (and DSV) bound simultaneously can have different areas mapped versus non-mapped and can have different sized surface formats (which means different tile shapes).

Here is the ideal behavior:

Reads from RTVs return 0 in missing tiles and writes are dropped. This ideal definition for write handling is not required by [Tier 2](#); writes to non-mapped tiles can end up in a cache that subsequent reads could pick up.

## Related topics

[Pipeline access to tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tile access limitations with duplicate mappings

Article • 08/23/2019

This section describes tile access limitations with duplicate mappings.

## Copying tiled resources with overlapping source and destination

If tiles in the source and destination area of a Copy\* operation have duplicated mappings in the copy area that would have overlapped even if both resources were not tiled resources and the Copy\* operation supports overlapping copies, the Copy\* operation will behave fine (as if the source is copied to a temporary location before going to the destination). But if the overlap is not obvious (like the source and destination resources are different but share mappings or mappings are duplicated over a given surface), results of the copy operation on the tiles that are shared are undefined.

## Copying to tiled resource with duplicated tiles in destination area

Copying to a tiled resource with duplicated tiles in the destination area produces undefined results in these tiles unless the data itself is identical; different tiles might write the tiles in different orders.

## UAV accesses to duplicate tiles mappings

Suppose an unordered access view (UAV) on a tiled resource has duplicate tile mappings in its area or with other resources bound to the pipeline. Ordering of accesses to these duplicated tiles is undefined if performed by different threads, just as any ordering of memory access to UAVs in general is unordered.

## Rendering after tile mapping changes or content updates from outside mappings

If a tiled resource's tile mappings have changed or content in mapped tiled pool tiles have changed via another tiled resource's mappings, and the tiled resource is going to

be rendered via render target view or depth stencil view, the application must Clear (using the fixed function Clear APIs) or fully copy over using Copy\*/Update\* APIs the tiles that have changed within the area being rendered (mapped or not). Failure of an application to clear or copy in these cases results in hardware optimization structures for the given render target view or depth stencil view being stale and will result in garbage rendering results on some hardware and inconsistency across different hardware. These hidden optimization data structures used by hardware might be local to individual mappings and not visible to other mappings to the same memory.

The [ID3D11DeviceContext1::ClearView](#) operation supports clearing render target views with rectangles. For hardware that supports tiled resources, **ClearView** must also support clearing of depth stencil views with rectangles, for depth only surfaces (without stencil). This operation allows applications to clear only the necessary area of a surface.

If an application needs to preserve existing memory contents of areas in a tiled resource where mappings have changed, that application must work around the clear requirement. The application can accomplish this work-around by first saving the contents where tile mappings have changed (by copying them to a temporary surface, for example, by using [ID3D11DeviceContext2::CopyTiles](#)), issuing the required clear command and then copying the contents back. While this would accomplish the task of preserving surface contents for incremental rendering, the downside is that subsequent rendering performance on the surface might suffer because rendering optimizations might be lost.

If a tile is mapped into multiple tiled resources at the same time and tile contents are manipulated by any means (render, copy, and so on) via one of the tiled resources, if the same tile is to be rendered via any other tiled resource, the tile must be cleared first as previously described.

## Rendering to tiles shared outside render area

Suppose an area in a tiled resource is being rendered to and the tile pool tiles referenced by the render area are also mapped to from outside the render area (including via other tiled resources, at the same time or not). Data rendered to these tiles isn't guaranteed to appear correctly when viewed through the other mappings, even though the underlying memory layout is compatible. This fact is due to optimization data structures some hardware use that can be local to individual mappings for renderable surfaces and not visible to other mappings to the same memory location. You can work around this restriction by copying from the rendered mapping to all the other mappings to the same memory that might be accessed (or clearing that memory or copying other data to it if the old contents are no longer

needed). While this work-around seems redundant, it makes all other mappings to the same memory correctly understand how to access its contents, and at least the memory savings of having only a single physical memory backing remains intact. Also, when you switch between using different tiled resources that share mappings (unless only reading), you must call the [ID3D11DeviceContext2::TiledResourceBarrier](#) API in between the switches.

## Rendering to tiles shared within render area

If an area in a tiled resource is being rendered to and within the render area multiple tiles are mapped to the same tile pool location, rendering results are undefined on those tiles.

## Data compatibility across tiled resources sharing tiles

Suppose multiple tiled resources have mappings to the same tile pool locations and each resource is used to access the same data. This scenario is only valid if the other rules about avoiding problems with hardware optimization structures are avoided, appropriate calls to [ID3D11DeviceContext2::TiledResourceBarrier](#) are made, and the tiled resources are compatible with each other. The latter is described here in terms of what it means for tiled resources sharing tiles to be incompatible. The incompatibility conditions of accessing the same data across duplicate tile mappings are the use of different surface dimensions or format, or differences in the presence of render target or depth stencil bind flags on the resources. Writing to the memory with one type of mapping produces undefined results if you subsequently read or render via a mapping from an incompatible resource. If the other resource sharing mappings are first initialized with new data (recycling the memory for a different purpose), the subsequent read or render operation is fine since data isn't bleeding across incompatible interpretations. But, you must call the [TiledResourceBarrier](#) API when you switch between accessing incompatible mappings like this.

If the render target or depth stencil bind flag isn't set on any of the resources sharing mappings with each other, there are far fewer restrictions. As long as the format and surface types (for example, Texture2D) are the same, tiles can be shared. Different formats being compatible are cases such as BC\* surfaces and the equivalent sized uncompressed 32 bit or 16 bit per component format, like BC6H and R32G32B32A32. Many 32 bit per element formats can be aliased with R32\_\* as well (R10G10B10A2\_\*, R8G8B8A8\_\*, B8G8R8A8\_\*, B8G8R8X8\_\*, R16G16\_\*); this operation has always been allowed for non-tiled resources.

Sharing between packed and non-packed tiles is fine if the formats are compatible and the tiles are filled with solid color.

Finally, if nothing is common about the resources sharing tile mappings except that none have render target or depth stencil bind flags, only memory filled with 0 can be shared safely; the mapping will appear as whatever 0 decodes to for the definition of the given resource format (typically just 0).

## Related topics

[Pipeline access to tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tiled resources texture sampling features

Article • 08/19/2020

This section describes tiled resources texture sampling features.

## Requirements of tiled resources texture sampling features

The texture sampling features described here require [Tier 2](#) level of tiled resources support.

## Shader feedback about mapped areas

Any shader instruction that reads and/or writes to a tiled resource causes status information to be recorded. This status is exposed as an optional extra return value on every resource access instruction that goes into a 32-bit temp register. The contents of the return value are opaque. That is, direct reading by the shader program is disallowed. But, you can use the [CheckAccessFullyMapped](#) function to extract the status info.

## Fully mapped check

The [CheckAccessFullyMapped](#) function interprets the status returned from a memory access and indicates whether all data being accessed was mapped in the resource. [CheckAccessFullyMapped](#) returns true (0xFFFFFFFF) if data was mapped or false (0x00000000) if data was unmapped.

During filter operations, sometimes the weight of a given texel ends up being 0.0. An example is a linear sample with texture coordinates that fall directly on a texel center: 3 other texels (which ones they are can vary by hardware) contribute to the filter but with 0 weight. These 0 weight texels don't contribute to the filter result at all, so if they happen to fall on **NULL** tiles, they don't count as an unmapped access. Note the same guarantee applies for texture filters that include multiple mip levels; if the texels on one of the mips isn't mapped but the weight on those texels is 0, those texels don't count as an unmapped access.

When sampling from a format that has fewer than 4 components (such as `DXGI_FORMAT_R8_UNORM`), any texels that fall on **NULL** tiles result in the a **NULL**

mapped access being reported regardless of which components the shader actually looks at in the result. For example, reading from R8\_UNORM and masking the read result in the shader with .gba/.yzw wouldn't appear to need to read the texture at all. But if the texel address is a **NULL** mapped tile, the operation still counts as a **NULL** map access.

The shader can check the status and pursue any desired course of action on failure. For example, a course of action can be logging 'misses' (say via UAV write) and/or issuing another read clamped to a coarser LOD known to be mapped. An application might want to track successful accesses as well in order to get a sense of what portion of the mapped set of tiles got accessed.

One complication for logging is no mechanism exists for reporting the exact set of tiles that would have been accessed. The application can make conservative guesses based on knowing the coordinates it used for access, as well as using the LOD instruction (for example, `tex2Dlod`) which returns what the hardware LOD calculation is.

Another complication is that lots of accesses will be to the same tiles, so a lot of redundant logging will occur and possibly contention on memory. It could be convenient if the hardware could be given the option to not bother to report tile accesses if they were reported elsewhere before. Perhaps the state of such tracking could be reset from the API (likely at frame boundaries).

## Per-sample MinLOD clamp

To help shaders avoid areas in mipmapped tiled resources that are known to be non-mapped, most shader instructions that involve using a sampler (filtering) have a new mode that allows the shader to pass an additional float32 MinLOD clamp parameter to the texture sample. This value is in the view's mipmap number space, as opposed to the underlying resource.

The hardware performs `max(fShaderMinLODClamp,fComputedLOD)` in the same place in the LOD calculation where the per-resource MinLOD clamp occurs, which is also a `max()`.

If the result of applying the per-sample LOD clamp and any other LOD clamps defined in the sampler is an empty set, the result is the same out of bounds access result as the per-resource minLOD clamp: 0 for components in the surface format and defaults for missing components.

The LOD instruction (for example, `tex2Dlod`), which predates the per-sample minLOD clamp described here, returns both a clamped and unclamped LOD. The clamped LOD returned from this LOD instruction reflects all clamping including the per-resource

clamp, but not a per-sample clamp. Per-sample clamp is controlled and known by the shader anyway, so the shader author can manually apply that clamp to the LOD instruction's return value if desired.

## Min/Max reduction filtering

Applications can choose to manage their own data structures that inform them of what the mappings looks like for a tiled resource. An example would be a surface that contains a texel to hold information for every tile in a tiled resource. One might store the first LOD that is mapped at a given tile location. By careful sampling of this data structure in a similar way that the tiled resource is intended to be sampled, one might discover what the minimum LOD that is fully mapped for an entire texture filter footprint will be. To help make this process easier, Direct3D 11.2 introduces a new general purpose sampler mode, min/max filtering.

The utility of min/max filtering for LOD tracking might be useful for other purposes, such as, perhaps the filtering of depth surfaces.

Min/max reduction filtering is a mode on samplers that fetches the same set of texels that a normal texture filter would fetch. But instead of blending the values to produce an answer, it returns the min() or max() of the texels fetched, on a per-component basis (for example, the min of all the R values, separately from the min of all the G values and so on).

The min/max operations follow Direct3D arithmetic precision rules. The order of comparisons doesn't matter.

During filter operations that aren't min/max, sometimes the weight of a given texel ends up being 0.0. An example is a linear sample with texture coordinates that fall directly on a texel center - 3 other texels (which ones they are may vary by hardware) contribute to the filter but with 0 weight. For any of these texels that would be 0 weight on a non-min/max filter, if the filter is min/max, these texels still do not contribute to the result (and the weights do not otherwise affect the min/max filter operation).

The full list of filter modes is shown in the [D3D11\\_FILTER](#) enumeration with MINIMUM and MAXIMUM in the enumeration values.

Support for this feature depends on [Tier 2](#) support for tiled resources.

## Related topics

[Pipeline access to tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# HLSL tiled resources exposure

Article • 08/19/2020

New Microsoft High Level Shader Language (HLSL) syntax is required to support tiled resources in [Shader Model 5](#).

The new HLSL syntax is allowed only on devices with tiled resources support. Each relevant HLSL method for tiled resources in the following table accepts either one (feedback) or two (clamp and feedback in this order) additional optional parameters. For example, a **Sample** method is:

**Sample(sampler, location [, offset [, clamp [, feedback] ] ])**

An example of a **Sample** method is [Texture2D.Sample\(S,float,int,float,uint\)](#).

The offset, clamp and feedback parameters are optional. You must specify all optional parameters up to the one you need, which is consistent with the C++ rules for default function arguments. For example, if the feedback status is needed, both offset and clamp parameters need to be explicitly supplied to **Sample**, even though they may not be logically needed.

The clamp parameter is a scalar float value. The literal value of clamp=0.0f indicates that the clamp operation isn't performed.

The feedback parameter is a **uint** variable that you can supply to the memory-access querying intrinsic [CheckAccessFullyMapped](#) function. You must not modify or interpret the value of the feedback parameter; but, the compiler doesn't provide any advanced analysis and diagnostics to detect whether you modified the value.

Here is the syntax of [CheckAccessFullyMapped](#):

```
bool CheckAccessFullyMapped(in uint FeedbackVar);
```

[CheckAccessFullyMapped](#) interprets the value of *FeedbackVar* and returns true if all data being accessed was mapped in the resource; otherwise, [CheckAccessFullyMapped](#) returns false.

If either the clamp or feedback parameter is present, the compiler emits a variant of the basic instruction. For example, sample of a tiled resource generates the `sample_c1_s` instruction. If neither clamp nor feedback is specified, the compiler emits the basic instruction, so that there is no change from the current behavior. The clamp value of 0.0f indicates that no clamp is performed; thus, the driver compiler can further tailor the instruction to the target hardware. If feedback is a NULL register in an instruction, the

feedback is unused; thus, the driver compiler can further tailor the instruction to the target architecture.

If the HLSL compiler infers that clamp is 0.0f and feedback is unused, the compiler emits the corresponding basic instruction (for example, `sample` rather than `sample_cl_s`).

If a tiled resource access consists of several constituent byte code instructions, for example, for structured resources, the compiler aggregates individual feedback values via the OR operation to produce the final feedback value. Therefore, you see a single feedback value for such a complex access.

This is the summary table of HLSL methods that are changed to support feedback and/or clamp. These all work on tiled and non-tiled resources of all dimensions. Non-tiled resources always appear to be fully mapped.

HLSL objects	Intrinsic methods with feedback option (*) - also has clamp option
[RW]Texture2D	Gather
[RW]Texture2DArray	GatherRed
TextureCUBE	GatherGreen
TextureCUBEArray	GatherBlue GatherAlpha GatherCmp GatherCmpRed GatherCmpGreen GatherCmpBlue GatherCmpAlpha
[RW]Texture1D	Sample*
[RW]Texture1DArray	SampleBias*
[RW]Texture2D	SampleCmp*
[RW]Texture2DArray	SampleCmpLevelZero
[RW]Texture3D	SampleGrad*
TextureCUBE	SampleLevel
TextureCUBEArray	
[RW]Texture1D	Load
[RW]Texture1DArray	
[RW]Texture2D	
Texture2DMS	
[RW]Texture2DArray	
Texture2DArrayMS	
[RW]Texture3D	
[RW]Buffer	
[RW]ByteAddressBuffer	
[RW]StructuredBuffer	

## Related topics

[Pipeline access to tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tiled resources features tiers

Article • 08/23/2019

Direct3D 11.2 exposes tiled resources support in two tiers with the [D3D11\\_TILED\\_RESOURCES\\_TIER](#) values.

To query whether the hardware and driver support tiled resources and at what tier level, pass the [D3D11\\_FEATURE\\_D3D11\\_OPTIONS1](#) value to the *Feature* parameter of [ID3D11Device::CheckFeatureSupport](#). Also, pass a pointer to the [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS1](#) structure to the *pFeatureSupportData* parameter, and pass the size of the [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS1](#) structure to the *FeatureSupportDataSize* parameter. [CheckFeatureSupport](#) returns the tier level as a [D3D11\\_TILED\\_RESOURCES\\_TIER](#) value in the *TiledResourcesTier* member of [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS1](#).

This section describes these two tiers.

## In this section

Topic	Description
<a href="#">Tier 1</a>	This section describes tier 1 support.
<a href="#">Tier 2</a>	This section describes tier 2 support.

## Related topics

[Tiled resources](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tier 1

Article • 08/19/2020

This section describes tier 1 support.

- Hardware at feature level 11.0 minimum.
- No quilting support.
- No Texture1D or Texture3D support.
- No 2, 8 or 16 sample multisample antialiasing (MSAA) support. Only 4x is required, except no 128 bpp formats.
- No standard swizzle pattern (layout within 64KB tiles and tail mip packing is up to the hardware vendor).
- Limitations on how tiles can be accessed when there are duplicate mappings, described in [Tile access limitations with duplicate mappings](#).

## Limitations affecting tier 1 only

- Tiled resources can have **NULL** mappings but reading from them or writing to them produces undefined results, including device removed. Applications can get around this by mapping a single dummy page to all the empty areas. Take care if you write and render to a page that is mapped to multiple render target locations because the order of writes will be undefined.
- Shader instructions for clamping LOD and mapped status feedback are not available. For more info, see [HLSL tiled resources exposure](#).
- Alignment constraints for standard tile shapes: It is only guaranteed that mips (starting from the finest) whose dimensions are all multiples of the standard tile size support the standard tile shapes and can have individual tiles arbitrarily mapped/unmapped. The first mipmap in a tiled resource that has any dimension not a multiple of standard tile size, along with all coarser mipmaps, can have a non-standard tiling shape, fitting into N 64KB tiles for this set of mips at once (N reported to the application). These N tiles are considered packed as one unit, which must be either fully mapped or fully unmapped by the application at any given time, though the mappings of each of the N tiles can be at arbitrarily disjoint locations in a tile pool.
- Tiled resources with any mipmaps not a multiple of standard tile size in all dimensions are not allowed to have an array size larger than 1.
- In order to switch between referencing tiles in a tile pool via a **Buffer** resource to referencing the same tiles via a **Texture** resource, or vice-versa, the most recent call to [UpdateTileMappings](#) or [CopyTileMappings](#) that defines mappings to those tile pool tiles must be for the same resource dimension (Buffer versus Texture\*) as the

resource dimension that will be used to access the tiles. Otherwise, behavior is undefined including the chance of device reset. So, for example, calling **UpdateTileMappings** to define tile mappings for a Buffer, then **UpdateTileMappings** to the same tiles in the tile pool via a **Texture2D** resource, then accessing the tiles via the Buffer is invalid. Work-around operations are to either redefine tile mappings for a resource when switching between Buffer and Texture (or vice versa) sharing tiles or just never sharing tiles in a tile pool between Buffer resources and Texture resources.

- Min/Max reduction filtering is not supported. For info about Min/Max reduction filtering, see [Tiled resources texture sampling features](#).

## Related topics

[Tiled resources features tiers](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tier 2

Article • 08/19/2020

This section describes tier 2 support.

- Hardware at Feature Level 11.1 minimum.
- All features of the previous tier (without [Tier 1](#) specific limitations) plus the additions in these following items:
- Shader instructions for clamping LOD and mapped status feedback are available. For more info, see [HLSL tiled resources exposure](#).
- Reads from non-mapped tiles return 0 in all non-missing components of the format, and the default for missing components.
- Writes to non-mapped tiles are stopped from going to memory but might end up in caches that subsequent reads to the same address might or might not pick up.
- Texture filtering with a footprint that straddles **NULL** and non-**NULL** tiles contributes 0 (with defaults for missing format components) for texels on **NULL** tiles into the overall filter operation. Some early hardware don't meet this requirement and returns 0 (with defaults for missing format components) for the full filter result if any texels (with nonzero weight) fall on a **NULL** tile. No other hardware will be allowed to miss the requirement to include all (nonzero weighted) texels in the filter operation.
- **NULL** texel accesses cause the [CheckAccessFullyMapped](#) operation on the status feedback for a texture read to return false. This is regardless of how the texture access result might get write masked in the shader and how many components are in the texture format (the combination of which might make it appear that the texture does not need to be accessed).
- Alignment constraints for standard tile shapes: Mipmaps that fill at least one standard tile in all dimensions are guaranteed to use the standard tiling, with the remainder considered packed as a **unit** into N tiles (N reported to the application). The application can map the N tiles into arbitrarily disjoint locations in a tile pool, but must either map all or none of the packed tiles. The mip packing is a unique set of packed tiles per array slice.
- Min/Max reduction filtering is supported. For info about Min/Max reduction filtering, see [Tiled resources texture sampling features](#).
- Tiled resources with any mipmaps less than standard tile size in any dimension are not allowed to have an array size larger than 1.
- Limitations on how tiles can be accessed when there are duplicate mappings, described in [Tile access limitations with duplicate mappings](#), continue to apply.

# Related topics

[Tiled resources](#) [features](#) [tiers](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Displayable surfaces

Article • 08/16/2024

Before displayable surfaces, presentation was generally done by creating a swap chain of buffers with identical properties, which were then cycled (flipped) through repeatedly, in order, to be presented to the screen. If you wanted to change the properties of a buffer to be presented, then you had to destroy that swap chain, and create a new one with all the buffers updated to the same new properties.

The displayable surfaces feature adds new operating system (OS) behavior that eliminates those restrictions (but it requires driver support in order to behave properly). Specifically, the feature means that buffers that are presented may have varying properties, and you may present them in any order.

The displayable surfaces (and flexible presentation) features, and their APIs, were introduced in Windows 11 (Build 10.0.22000.194). The functionality is enabled on supported drivers, beginning with WDDM 3.0 drivers, enabling enhanced presentation scenarios for Direct3D 11.

## Check for support, and use displayable surfaces

To determine whether the displayable surfaces feature is available on a system, call [ID3D11Device::CheckFeatureSupport](#). Pass [D3D11\\_FEATURE::D3D11\\_FEATURE\\_DISPLAYABLE](#), and receive a [D3D11\\_FEATURE\\_DATA\\_DISPLAYABLE](#) structure.

The [ID3D11Device::CreateTexture2D](#) API supports [D3D11\\_RESOURCE\\_MISC\\_FLAG::D3D11\\_RESOURCE\\_MISC\\_SHARED\\_DISPLAYABLE](#), which you can use in the [D3D11\\_TEXTURE2D\\_DESC::MiscFlags](#) member of the structure that you pass to [CreateTexture2D](#) in the *pDesc* parameter.

Textures with [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_DISPLAYABLE](#) are restricted to an array size of 1, and to 1 mip level.

When you use the [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_DISPLAYABLE](#) flag on the texture, you can show the texture on any active output (including multiple outputs simultaneously). Depending on the scenario, the texture might end up being consumed by the compositor (DWM), scanned out, or bound to various parts of the pipeline—potentially all simultaneously. For example, a capture texture from a camera might be shown on two displays, and a thumbnail of it shown on a third display, all at the same time—and all from the same allocation with no additional copies. In the case where a

displayable surface is to be scanned out on multiple displays, the OS will coordinate the collection of flip completes from the involved outputs before alerting your application that the surface is released back to it—no coordination of flip completion is required from the driver.

To present a displayable texture to the screen, you can use the [Composition Swapchain API](#). By using displayable surfaces rather than plain surfaces, the system is able to optimize presentation in some situations to bypass the system compositor and scanout the surfaces directly which reduces GPU/CPU overhead as well as overall latency. This is similar to using DXGI swapchains with the "flip" presentation modes. See [For best performance, use DXGI flip model](#) for more information.

Such textures as described above must be displayable for flexible presentation use. These textures are not required to have the same properties—for example, formats and sizes can differ, and these textures must be able to be displayed in arbitrary order ("out-of-order presentation"). Presentation will occur using the existing [Present1 DDI](#), with its existing calling patterns. For example, consider a pool of six buffers, three that are 720p (A, B, and C) and three that are 4K (D, E, and F): a valid presentation order could be A->E->C->B->F->E->D->C.

## Formats

The [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_DISPLAYABLE](#) flag is supported for the following formats in Direct3D 11:

- [DXGI\\_FORMAT::DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM](#)
- [DXGI\\_FORMAT\\_R8G8B8A8\\_UNORM](#)
- [DXGI\\_FORMAT\\_R16G16B16A16\\_FLOAT](#)
- [DXGI\\_FORMAT\\_R10G10B10A2\\_UNORM](#)
- [DXGI\\_FORMAT\\_NV12](#)
- [DXGI\\_FORMAT\\_YUY2](#)

A driver might also, optionally, support the following formats:

- [DXGI\\_FORMAT\\_P010](#)

You can use the following code example to check for displayable surfaces support for the optional formats above. The example involves calling [ID3D11Device::CheckFeatureSupport](#), and checking for [D3D11\\_FEATURE\\_FORMAT\\_SUPPORT2](#).

```
D3D11_FEATURE_DATA_FORMAT_SUPPORT2 FormatSupport2;
FormatSupport2.InFormat = DXGI_FORMAT_P010;
if (SUCCEEDED (hr = GetDevice())-
>CheckFeatureSupport(D3D11_FEATURE_FORMAT_SUPPORT2, &FormatSupport2,
sizeof(FormatSupport2)))
{
    if (FormatSupport2.OutFormatSupport2 &
D3D11_FORMAT_SUPPORT2_DISPLAYABLE)
    {
        // optional displayable format is supported
    }
}
```

## Flags

Shareable formats already generally support the following bind flags:

[D3D11\\_BIND\\_FLAG::D3D11\\_BIND\\_SHADER\\_RESOURCE](#),  
[D3D11\\_BIND\\_UNORDERED\\_ACCESS](#), [D3D11\\_BIND\\_RENDER\\_TARGET](#), and  
[D3D11\\_BIND\\_DECODER](#).

Existing supported usages of shared resources with the [D3D11\\_BIND\\_VIDEO\\_ENCODER](#) flag are extended to also support the [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_DISPLAYABLE](#) flag being added in these cases. Existing restrictions related to use of shared resources with [D3D11\\_BIND\\_VIDEO\\_ENCODER](#) are maintained.

[D3D11\\_BIND\\_VIDEO\\_ENCODER](#) and [D3D11\\_BIND\\_SHADER\\_RESOURCE](#) were previously mutually exclusive, except when combined with certain other bind flags. The exception has been extended to allow [D3D11\\_BIND\\_VIDEO\\_ENCODER](#) and [D3D11\\_BIND\\_SHADER\\_RESOURCE](#) to be used together when [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_DISPLAYABLE](#) is used.

The [D3D11\\_RESOURCE\\_MISC\\_HW\\_PROTECTED](#) flag is supported with the [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_DISPLAYABLE](#) flag.

## Related topics

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

---

## Feedback

Was this page helpful?

 Yes

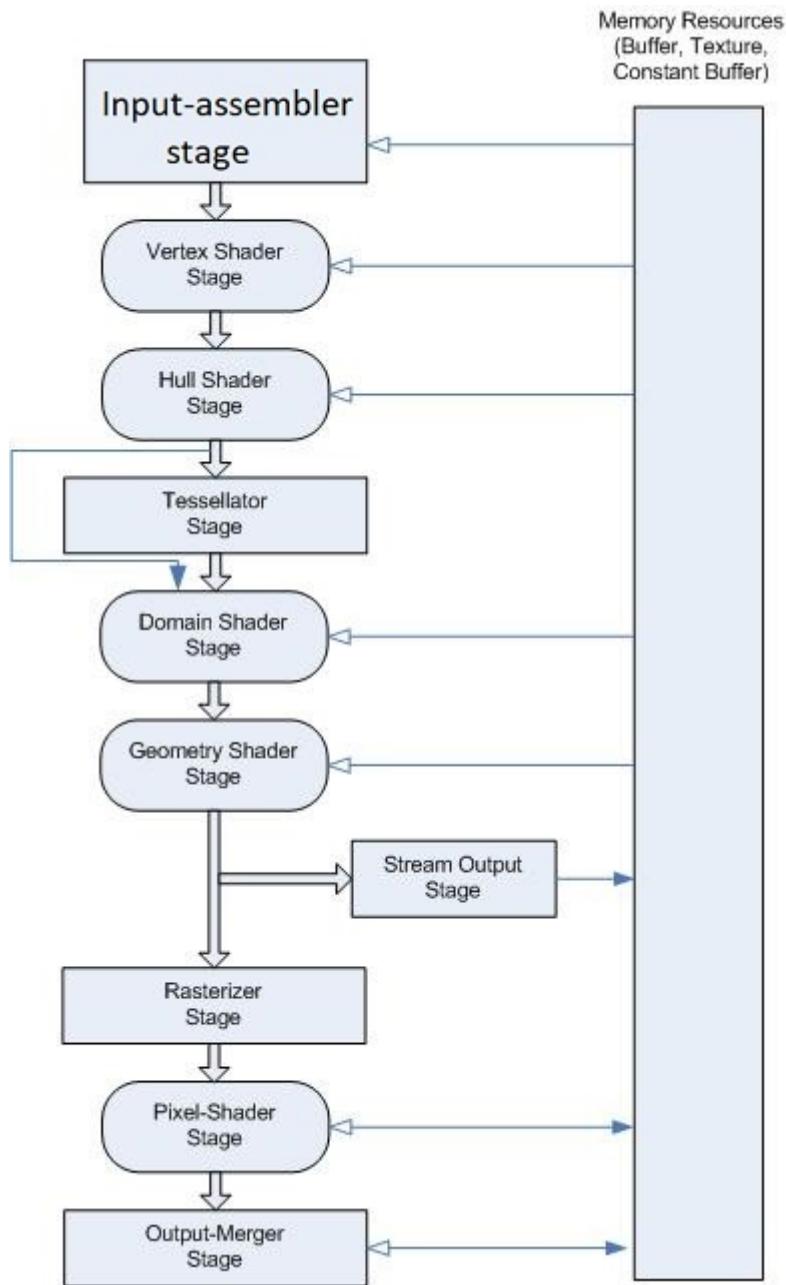
 No

Provide product feedback  | Get help at Microsoft Q&A

# Graphics pipeline

Article • 02/24/2022

The Direct3D 11 programmable pipeline is designed for generating graphics for realtime gaming applications. This section describes the Direct3D 11 programmable pipeline. The following diagram shows the data flow from input to output through each of the programmable stages.



The graphics pipeline for Microsoft Direct3D 11 supports the same stages as the [Direct3D 10 graphics pipeline](#), with additional stages to support advanced features.

You can use the Direct3D 11 API to configure all of the stages. Stages that feature common shader cores (the rounded rectangular blocks) are programmable by using the

HLSL programming language. As you will see, this makes the pipeline extremely flexible and adaptable.

## In this section

Topic	Description
Input-assembler stage	The Direct3D 10 and higher API separates functional areas of the pipeline into stages; the first stage in the pipeline is the input-assembler (IA) stage.
Vertex shader stage	The vertex-shader (VS) stage processes vertices from the input assembler, performing per-vertex operations such as transformations, skinning, morphing, and per-vertex lighting. Vertex shaders always operate on a single input vertex and produce a single output vertex. The vertex shader stage must always be active for the pipeline to execute. If no vertex modification or transformation is required, a pass-through vertex shader must be created and set to the pipeline.
Tessellation stages	The Direct3D 11 runtime supports three new stages that implement tessellation, which converts low-detail subdivision surfaces into higher-detail primitives on the GPU. Tessellation tiles (or breaks up) high-order surfaces into suitable structures for rendering.
Geometry shader stage	The geometry-shader (GS) stage runs application-specified shader code with vertices as input and the ability to generate vertices on output.
Stream-output stage	The purpose of the stream-output stage is to continuously output (or stream) vertex data from the geometry-shader stage (or the vertex-shader stage if the geometry-shader stage is inactive) to one or more buffers in memory (see <a href="#">Getting Started with the stream-output stage</a> ).
Rasterizer stage	The rasterization stage converts vector information (composed of shapes or primitives) into a raster image (composed of pixels) for the purpose of displaying real-time 3D graphics.
Pixel shader stage	The pixel-shader stage (PS) enables rich shading techniques such as per-pixel lighting and post-processing. A pixel shader is a program that combines constant variables, texture data, interpolated per-vertex values, and other data to produce per-pixel outputs. The rasterizer stage invokes a pixel shader once for each pixel covered by a primitive, however, it is possible to specify a <b>NULL</b> shader to avoid running a shader.
Output-merger stage	The output-merger (OM) stage generates the final rendered pixel color using a combination of pipeline state, the pixel data generated by the pixel shaders, the contents of the render targets, and the contents of the depth/stencil buffers. The OM stage is the final step for determining which pixels are visible (with depth-stencil testing) and blending the final pixel colors.

## Related topics

- [Compute shader](#)
  - [Programming guide for Direct3D 11](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

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

# Input-Assembler Stage

Article • 11/04/2020

The Direct3D 10 and higher API separates functional areas of the pipeline into stages; the first stage in the pipeline is the input-assembler (IA) stage.

The purpose of the input-assembler stage is to read primitive data (points, lines and/or triangles) from user-filled buffers and assemble the data into primitives that will be used by the other pipeline stages. The IA stage can assemble vertices into several different [primitive types](#) (such as line lists, triangle strips, or primitives with adjacency). New primitive types (such as a line list with adjacency or a triangle list with adjacency) have been added to support the geometry shader.

Adjacency information is visible to an application only in a geometry shader. If a geometry shader were invoked with a triangle including adjacency, for instance, the input data would contain 3 vertices for each triangle and 3 vertices for adjacency data per triangle.

When the input-assembler stage is requested to output adjacency data, the input data must include adjacency data. This may require providing a dummy vertex (forming a degenerate triangle), or perhaps by flagging in one of the vertex attributes whether the vertex exists or not. This would also need to be detected and handled by a geometry shader, although culling of degenerate geometry will happen in the rasterizer stage.

While assembling primitives, a secondary purpose of the IA is to attach [system-generated values](#) to help make shaders more efficient. System-generated values are text strings that are also called semantics. All three shader stages are constructed from a common shader core, and the shader core uses system-generated values (such as a primitive id, an instance id, or a vertex id) so that a shader stage can reduce processing to only those primitives, instances, or vertices that have not already been processed.

As shown in the [pipeline-block diagram](#), once the IA stage reads data from memory (assembles the data into primitives and attaches system-generated values), the data is output to the [vertex shader stage](#).

## In this section

Topic	Description

Topic	Description
<a href="#">Getting Started with the Input-Assembler Stage</a>	There are a few steps necessary to initialize the input-assembler (IA) stage. For example, you need to create buffer resources with the vertex data that the pipeline needs, tell the IA stage where the buffers are and what type of data they contain, and specify the type of primitives to assemble from the data.
<a href="#">Primitive Topologies</a>	Direct3D 10 and higher supports several primitive types (or topologies) that are represented by the <a href="#">D3D_PRIMITIVE_TOPOLOGY</a> enumerated type. These types define how vertices are interpreted and rendered by the pipeline.
<a href="#">Using the Input-Assembler Stage without Buffers</a>	Creating and binding buffers is not necessary if your shaders don't require buffers. This section contains an example of simple vertex and pixel shaders that draw a single triangle.
<a href="#">Using System-Generated Values</a>	System-generated values are generated by the IA stage (based on user-supplied input <a href="#">semantics</a> ) to allow certain efficiencies in shader operations.

## Related topics

[Graphics Pipeline](#)

[Pipeline Stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Getting Started with the Input-Assembler Stage

Article • 08/19/2020

There are a few steps necessary to initialize the input-assembler (IA) stage. For example, you need to create buffer resources with the vertex data that the pipeline needs, tell the IA stage where the buffers are and what type of data they contain, and specify the type of primitives to assemble from the data.

The basic steps involved in setting up the IA stage, shown in the following table, are covered in this topic.

Step	Description
Create Input Buffers	Create and initialize input buffers with input vertex data.
Create the Input-Layout Object	Define how the vertex buffer data will be streamed into the IA stage by using an input-layout object.
Bind Objects to the Input-Assembler Stage	Bind the created objects (input buffers and the input-layout object) to the IA stage.
Specify the Primitive Type	Identify how the vertices will be assembled into primitives.
Call Draw Methods	Send the data bound to the IA stage through the pipeline.

After you understand these steps, move on to [Using System-Generated Values](#).

## Create Input Buffers

There are two types of input buffers: [vertex buffers](#) and index buffers. Vertex buffers supply vertex data to the IA stage. Index buffers are optional; they provide indices to vertices from the vertex buffer. You may create one or more vertex buffers and, optionally, an index buffer.

After you create the buffer resources, you need to create an input-layout object to describe the data layout to the IA stage, and then you need to bind the buffer resources to the IA stage. Creating and binding buffers is not necessary if your shaders don't use buffers. For an example of a simple vertex and pixel shader that draws a single triangle, see [Using the Input-Assembler Stage without Buffers](#).

For help with creating a vertex buffer, see [Create a Vertex Buffer](#). For help with creating an index buffer, see [Create an Index Buffer](#).

## Create the Input-Layout Object

The input-layout object encapsulates the input state of the IA stage. This includes a description of the input data that is bound to the IA stage. The data is streamed into the IA stage from memory, from one or more vertex buffers. The description identifies the input data that is bound from one or more vertex buffers and gives the runtime the ability to check the input data types against the shader input parameter types. This type checking not only verifies that the types are compatible, but also that each of the elements that the shader requires is available in the buffer resources.

An input-layout object is created from an array of input-element descriptions and a pointer to the compiled shader (see [ID3D11Device::CreateInputLayout](#)). The array contains one or more input elements; each input element describes a single vertex-data element from a single vertex buffer. The entire set of input-element descriptions describes all of the vertex-data elements from all of the vertex buffers that will be bound to the IA stage.

The following layout description describes a single vertex buffer that contains three vertex-data elements:

```
D3D11_INPUT_ELEMENT_DESC layout[] =
{
    { L"POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
        D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { L"TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, 12,
        D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { L"NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 20,
        D3D11_INPUT_PER_VERTEX_DATA, 0 },
};
```

An input-element description describes each element contained by a single vertex in a vertex buffer, including size, type, location, and purpose. Each row identifies the type of data by using the semantic, the semantic index, and the data format. A [semantic](#) is a text string that identifies how the data will be used. In this example, the first row identifies 3-component position data (*xyz*, for example); the second row identifies 2-component texture data (*UV*, for example); and the third row identifies normal data.

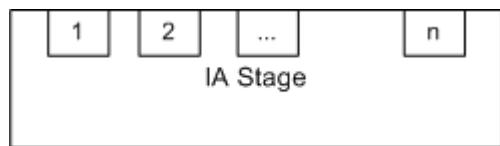
In this example of an input-element description, the semantic index (which is the second parameter) is set to zero for all three rows. The semantic index helps distinguish

between two rows that use the same semantics. Since there are no similar semantics in this example, the semantic index can be set to its default value, zero.

The third parameter is the *format*. The format (see [DXGI\\_FORMAT](#)) specifies the number of components per element, and the data type, which defines the size of the data for each element. The format can be fully typed at the time of resource creation, or you may create a resource by using a [DXGI\\_FORMAT](#), which identifies the number of components in an element, but leaves the data type undefined.

## Input Slots

Data enters the IA stage through inputs called *input slots*, as shown in the following illustration. The IA stage has  $n$  input slots, which are designed to accommodate up to  $n$  vertex buffers that provide input data. Each vertex buffer must be assigned to a different slot; this information is stored in the input-layout declaration when the input-layout object is created. You may also specify an offset from the start of each buffer to the first element in the buffer to be read.



The next two parameters are the *input slot* and the *input offset*. When you use multiple buffers, you can bind them to one or more input slots. The input offset is the number of bytes between the start of the buffer and the beginning of the data.

## Reusing Input-Layout Objects

Each input-layout object is created based on a shader signature; this allows the API to validate the input-layout-object elements against the shader-input signature to make sure that there is an exact match of types and semantics. You can create a single input-layout object for many shaders, as long as all of the shader-input signatures exactly match.

## Bind Objects to the Input-Assembler Stage

After you create vertex buffer resources and an input layout object, you can bind them to the IA stage by calling [ID3D11DeviceContext::IASetVertexBuffers](#) and [ID3D11DeviceContext::IASetInputLayout](#). The following example shows the binding of a single vertex buffer and an input-layout object to the IA stage:

```

UINT stride = sizeof( SimpleVertex );
UINT offset = 0;
g_pd3dDevice->IASetVertexBuffers(
    0,                  // the first input slot for binding
    1,                  // the number of buffers in the array
    &g_pVertexBuffer, // the array of vertex buffers
    &stride,           // array of stride values, one for each buffer
    &offset );         // array of offset values, one for each buffer

// Set the input layout
g_pd3dDevice->IASetInputLayout( g_pVertexLayout );

```

Binding the input-layout object only requires a pointer to the object.

In the preceding example, a single vertex buffer is bound; however, multiple vertex buffers can be bound by a single call to [ID3D11DeviceContext::IASetVertexBuffers](#), and the following code shows such a call to bind three vertex buffers:

```

UINT strides[3];
strides[0] = sizeof(SimpleVertex1);
strides[1] = sizeof(SimpleVertex2);
strides[2] = sizeof(SimpleVertex3);
UINT offsets[3] = { 0, 0, 0 };
g_pd3dDevice->IASetVertexBuffers(
    0,                  //first input slot for binding
    3,                  //number of buffers in the array
    &g_pVertexBuffer, //array of three vertex buffers
    &strides,           //array of stride values, one for each buffer
    &offsets );         //array of offset values, one for each buffer

```

An index buffer is bound to the IA stage by calling [ID3D11DeviceContext::IASetIndexBuffer](#).

## Specify the Primitive Type

After the input buffers have been bound, the IA stage must be told how to assemble the vertices into primitives. This is done by specifying the [primitive type](#) by calling [ID3D11DeviceContext::IASetPrimitiveTopology](#); the following code calls this function to define the data as a triangle list without adjacency:

```

g_pd3dDevice->IASetPrimitiveTopology( D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST );

```

The rest of the primitive types are listed in [D3D\\_PRIMITIVE\\_TOPOLOGY](#).

## Call Draw Methods

After input resources have been bound to the pipeline, an application calls a draw method to render primitives. There are several draw methods, which are shown in the following table; some use index buffers, some use instance data, and some reuse data from the streaming-output stage as input to the input-assembler stage.

Draw Methods	Description
<a href="#">ID3D11DeviceContext::Draw</a>	Draw non-indexed, non-instanced primitives.
<a href="#">ID3D11DeviceContext::DrawInstanced</a>	Draw non-indexed, instanced primitives.
<a href="#">ID3D11DeviceContext::DrawIndexed</a>	Draw indexed, non-instanced primitives.
<a href="#">ID3D11DeviceContext::DrawIndexedInstanced</a>	Draw indexed, instanced primitives.
<a href="#">ID3D11DeviceContext::DrawAuto</a>	Draw non-indexed, non-instanced primitives from input data that comes from the streaming-output stage.

Each draw method renders a single topology type. During rendering, incomplete primitives (those without enough vertices, lacking indices, partial primitives, and so on) are silently discarded.

## Related topics

[Input-Assembler Stage](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Primitive Topologies

Article • 08/19/2020

Direct3D 10 and higher supports several primitive types (or topologies) that are represented by the [D3D\\_PRIMITIVE\\_TOPOLOGY](#) enumerated type. These types define how vertices are interpreted and rendered by the pipeline.

- [Basic Primitive Types](#)
- [Primitive Adjacency](#)
- [Winding Direction and Leading Vertex Positions](#)
- [Generating Multiple Strips](#)
- [Related topics](#)

## Basic Primitive Types

The following basic primitive types are supported:

- [Point List](#)
- [Line List](#)
- [Line Strip](#)
- [Triangle List](#)
- [Triangle Strip](#)

For a visualization of each primitive type, see the diagram later in this topic in [Winding Direction and Leading Vertex Positions](#).

The input-assembler stage reads data from vertex and index buffers, assembles the data into these primitives, and then sends the data to the remaining pipeline stages. (You can use the [ID3D11DeviceContext::IASetPrimitiveTopology](#) method to specify the primitive type for the input-assembler stage.)

## Primitive Adjacency

All Direct3D 10 and higher primitive types (except the point list) are available in two versions: one primitive type with adjacency and one primitive type without adjacency. Primitives with adjacency contain some of the surrounding vertices, while primitives without adjacency contain only the vertices of the target primitive. For example, the line list primitive (represented by the [D3D\\_PRIMITIVE\\_TOPOLOGY\\_LINELIST](#) value) has a corresponding line list primitive that includes adjacency (represented by the [D3D\\_PRIMITIVE\\_TOPOLOGY\\_LINELIST\\_ADJ](#) value.)

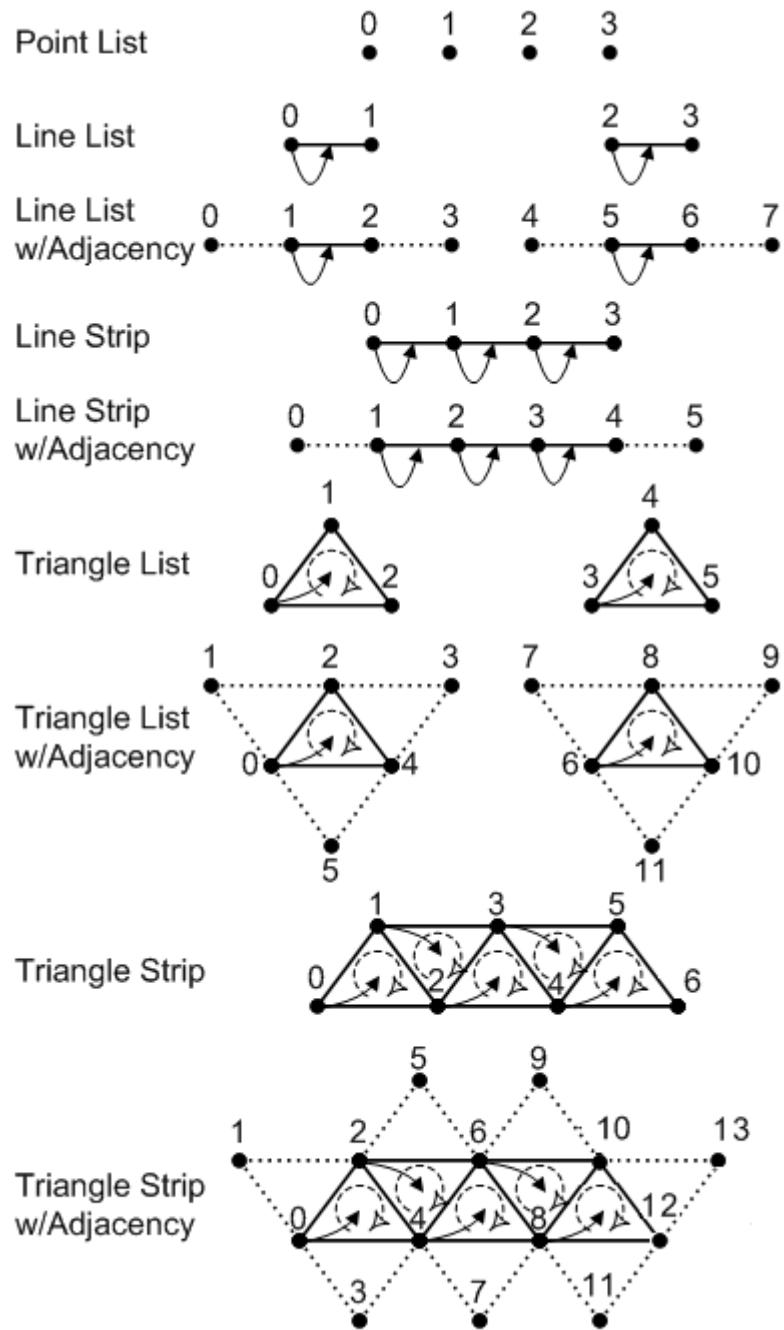
Adjacent primitives are intended to provide more information about your geometry and are only visible through a geometry shader. Adjacency is useful for geometry shaders that use silhouette detection, shadow volume extrusion, and so on.

For example, suppose you want to draw a triangle list with adjacency. A triangle list that contains 36 vertices (with adjacency) will yield 6 completed primitives. Primitives with adjacency (except line strips) contain exactly twice as many vertices as the equivalent primitive without adjacency, where each additional vertex is an adjacent vertex.

## Winding Direction and Leading Vertex Positions

As shown in the following illustration, a leading vertex is the first non-adjacent vertex in a primitive. A primitive type can have multiple leading vertices defined, as long as each one is used for a different primitive. For a triangle strip with adjacency, the leading vertices are 0, 2, 4, 6, and so on. For a line strip with adjacency, the leading vertices are 1, 2, 3, and so on. An adjacent primitive, on the other hand, has no leading vertex.

The following illustration shows the vertex ordering for all of the primitive types that the input assembler can produce.



The symbols in the preceding illustration are described in the following table.

Symbol	Name	Description
●	Vertex	A point in 3D space.
○ ↗	Winding Direction	The vertex order when assembling a primitive. Can be clockwise or counter-clockwise; specify this by calling <a href="#">ID3D11Device1::CreateRasterizerState1</a> .
↗	Leading Vertex	The first non-adjacent vertex in a primitive that contains per-constant data.

# Generating Multiple Strips

You can generate multiple strips through strip cutting. You can perform a strip cut by explicitly calling the [RestartStrip](#) HLSL function, or by inserting a special index value into the index buffer. This value is  $-1$ , which is `0xffffffff` for 32-bit indices or `0xffff` for 16-bit indices. An index of  $-1$  indicates an explicit 'cut' or 'restart' of the current strip. The previous index completes the previous primitive or strip and the next index starts a new primitive or strip. For more info about generating multiple strips, see [Geometry-Shader Stage](#).

 **Note**

You need **feature level 10.0** or higher hardware because not all 10level9 hardware implements this functionality.

## Related topics

[Getting Started with the Input-Assembler Stage](#)

[Pipeline Stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Using the Input-Assembler Stage without Buffers

Article • 08/23/2019

Creating and binding buffers is not necessary if your shaders don't require buffers. This section contains an example of simple vertex and pixel shaders that draw a single triangle.

- [Vertex Shader](#)
- [Pixel Shader](#)
- [Technique](#)
- [Application Code](#)
- [Related topics](#)

## Vertex Shader

For example, you could declare a vertex shader that creates its own vertices.

```
struct VSIn
{
    uint vertexId : SV_VertexID;
};

struct VSOut
{
    float4 pos : SV_Position;
    float4 color : color;
};

VSOut VSmain(VSIn input)
{
    VSOut output;

    if (input.vertexId == 0)
        output.pos = float4(0.0, 0.5, 0.5, 1.0);
    else if (input.vertexId == 2)
        output.pos = float4(0.5, -0.5, 0.5, 1.0);
    else if (input.vertexId == 1)
        output.pos = float4(-0.5, -0.5, 0.5, 1.0);

    output.color = clamp(output.pos, 0, 1);

    return output;
}
```

# Pixel Shader

```
// NoBuffer.fx
// Copyright (c) 2004 Microsoft Corporation. All rights reserved.
//

struct PSIn
{
    float4 pos : SV_Position;
    linear float4 color : color;
};

struct PSOut
{
    float4 color : SV_Target;
};

PSOut PSmain(PSIn input)
{
    PSOut output;

    output.color = input.color;

    return output;
}
```

# Technique

A technique is a collection of rendering passes (there must be at least one pass).

```
VertexShader vsCompiled = CompileShader( vs_4_0, VSmain() );

technique10 t0
{
    pass p0
    {
        SetVertexShader( vsCompiled );
        SetGeometryShader( NULL );
        SetPixelShader( CompileShader( ps_4_0, PSmain() ) );
    }
}
```

# Application Code

```
m_pD3D11Device->IASetInputLayout( NULL );  
  
m_pD3D11Device->IASetPrimitiveTopology( D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST );  
  
ID3DX11EffectTechnique * pTech = NULL;  
pTech = m_pEffect->GetTechniqueByIndex(0);  
pTech->GetPassByIndex(iPass)->Apply(0);  
  
m_pD3D11Device->Draw( 3, 0 );
```

## Related topics

[Getting Started with the Input-Assembler Stage](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Using System-Generated Values

Article • 05/24/2021

System-generated values are generated by the IA stage (based on user-supplied input [semantics](#)) to allow certain efficiencies in shader operations.

By attaching data, such as an instance id (visible to VS), a vertex id (visible to VS), or a primitive id (visible to GS/PS), a subsequent shader stage may look for these system values to optimize processing in that stage. For instance, the VS stage may look for the instance id to grab additional per-vertex data for the shader or to perform other operations; the GS and PS stages may use the primitive id to grab per-primitive data in the same way.

- [VertexID](#)
- [PrimitiveID](#)
- [InstanceId](#)
- [Example](#)
- [Related topics](#)

## VertexID

A vertex id is used by each shader stage to identify each vertex. It is a 32-bit unsigned integer whose default value is 0. It is assigned to a vertex when the primitive is processed by the IA stage. Attach the vertex-id semantic to the shader input declaration to inform the IA stage to generate a per-vertex id.

The IA will add a vertex id to each vertex for use by shader stages. For each draw call, the vertex id is incremented by 1. Across indexed draw calls, the count resets back to the start value. For [ID3D11DeviceContext::DrawIndexed](#) and [ID3D11DeviceContext::DrawIndexedInstanced](#), the vertex id represents the index value. If the vertex id overflows (exceeds  $2^{32} - 1$ ), it wraps to 0.

For all primitive types, vertices have a vertex id associated with them (regardless of adjacency).

## PrimitiveID

A primitive id is used by each shader stage to identify each primitive. It is a 32-bit unsigned integer whose default value is 0. It is assigned to a primitive when the

primitive is processed by the IA stage. To inform the IA stage to generate a primitive id, attach the primitive-id semantic to the shader input declaration.

The IA stage will add a primitive id to each primitive for use by the geometry shader or the pixel shader stage (whichever is the first stage active after the IA stage). For each indexed draw call, the primitive id is incremented by 1, however, the primitive id resets to 0 whenever a new instance begins. All other draw calls do not change the value of the instance id. If the instance id overflows (exceeds  $2^{32} - 1$ ), it wraps to 0.

The pixel shader stage does not have a separate input for a primitive id; however, any pixel shader input that specifies a primitive id uses a constant interpolation mode.

There is no support for automatically generating a primitive id for adjacent primitives. For primitive types with adjacency, such as a triangle strip with adjacency, a primitive id is only maintained for the interior primitives (the non-adjacent primitives), just like the set of primitives in a triangle strip without adjacency.

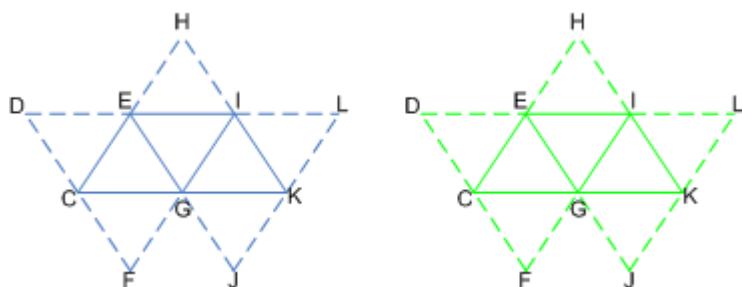
## InstanceID

An instance id is used by each shader stage to identify the instance of the geometry that is currently being processed. It is a 32-bit unsigned integer whose default value is 0.

The IA stage will add an instance id to each vertex if the vertex shader input declaration includes the instance id semantic. For each indexed draw call, instance id is incremented by 1. All other draw calls do not change the value of instance id. If instance id overflows (exceeds  $2^{32} - 1$ ), it wraps to 0.

## Example

The following illustration shows how system values are attached to an instanced triangle strip in the IA stage.



These tables show the system values generated for both instances of the same triangle strip. The first instance (instance U) is shown in blue, the second instance (instance V) is

shown in green. The solid lines connect the vertices in the primitives, the dashed lines connect the adjacent vertices.

The following tables show the system-generated values for the instance U.

<b>Vertex Data</b>	<b>C,U</b>	<b>D,U</b>	<b>E,U</b>	<b>F,U</b>	<b>G,U</b>	<b>H,U</b>	<b>I,U</b>	<b>J,U</b>	<b>K,U</b>	<b>L,U</b>
<b>VertexID</b>	0	1	2	3	4	5	6	7	8	9
<b>InstanceID</b>	0	0	0	0	0	0	0	0	0	0

	<b>Value</b>	<b>Value</b>	<b>Value</b>
<b>PrimitiveID</b>	0	1	2
<b>InstanceID</b>	0	0	0

The following tables show the system-generated values for the instance V.

<b>Vertex Data</b>	<b>C,V</b>	<b>D,V</b>	<b>E,V</b>	<b>F,V</b>	<b>G,V</b>	<b>H,V</b>	<b>I,V</b>	<b>J,V</b>	<b>K,V</b>	<b>L,V</b>
<b>VertexID</b>	0	1	2	3	4	5	6	7	8	9
<b>InstanceID</b>	1	1	1	1	1	1	1	1	1	1

	<b>Value</b>	<b>Value</b>	<b>Value</b>
<b>PrimitiveID</b>	0	1	2
<b>InstanceID</b>	1	1	1

The input assembler generates the ids (vertex, primitive, and instance); notice also that each instance is given a unique instance id. The data ends with the strip cut, which separates each instance of the triangle strip.

## Related topics

[Input-Assembler Stage](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Vertex-shader stage

Article • 03/01/2022

The vertex-shader (VS) stage processes vertices from the input assembler, performing per-vertex operations such as transformations, skinning, morphing, and per-vertex lighting. Vertex shaders always operate on a single input vertex and produce a single output vertex. The vertex-shader stage must always be active for the pipeline to execute. If no vertex modification or transformation is required, a pass-through vertex shader must be created and set to the pipeline.

## The Vertex-shader

Each vertex shader input vertex can be comprised of up to 16 32-bit vectors (up to 4 components each) and each output vertex can be comprised of as many as 16 32-bit 4-component vectors. All vertex-shaders must have a minimum of one input and one output, which can be as little as one scalar value.

The vertex-shader stage can consume two system generated values from the input assembler: `VertexID` and `InstanceId` (see [System-value semantics](#)). Since `VertexID` and `InstanceId` are both meaningful at a vertex level, and IDs generated by hardware can only be fed into the first stage that understands them, these ID values can only be fed into the vertex-shader stage.

Vertex shaders are always run on all vertices, including adjacent vertices in input primitive topologies with adjacency. The number of times that the vertex-shader has been executed can be queried from the CPU using the `VSIInvocations` pipeline statistic.

A vertex-shader can perform load and texture sampling operations where screen-space derivatives are not required (using HLSL intrinsic functions: [Sample \(DirectX HLSL Texture Object\)](#), [SampleCmpLevelZero \(DirectX HLSL Texture Object\)](#), and [SampleGrad \(DirectX HLSL Texture Object\)](#)).

## Related topics

- [Graphics pipeline](#)
- [Pipeline stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Tessellation Stages

Article • 09/16/2020

The Direct3D 11 runtime supports three new stages that implement tessellation, which converts low-detail subdivision surfaces into higher-detail primitives on the GPU.

Tessellation tiles (or breaks up) high-order surfaces into suitable structures for rendering.

By implementing tessellation in hardware, a graphics pipeline can evaluate lower detail (lower polygon count) models and render in higher detail. While software tessellation can be done, tessellation implemented by hardware can generate an incredible amount of visual detail (including support for displacement mapping) without adding the visual detail to the model sizes and paralyzing refresh rates.

- [Tessellation Benefits](#)
- [New Pipeline Stages](#)
  - [Hull-Shader Stage](#)
  - [Tessellator Stage](#)
  - [Domain-Shader Stage](#)
- [APIs for initializing Tessellation Stages](#)
- [How to's:](#)
- [Related topics](#)

## Tessellation Benefits

Tessellation:

- Saves lots of memory and bandwidth, which allows an application to render higher detailed surfaces from low-resolution models. The tessellation technique implemented in the Direct3D 11 pipeline also supports displacement mapping, which can produce stunning amounts of surface detail.
- Supports scalable-rendering techniques, such as continuous or view dependent levels-of-detail which can be calculated on the fly.
- Improves performance by performing expensive computations at lower frequency (doing calculations on a lower-detail model). This could include blending calculations using blend shapes or morph targets for realistic animation or physics calculations for collision detection or soft body dynamics.

The Direct3D 11 pipeline implements tessellation in hardware, which off-loads the work from the CPU to the GPU. This can lead to very large performance improvements if an application implements large numbers of morph targets and/or more sophisticated

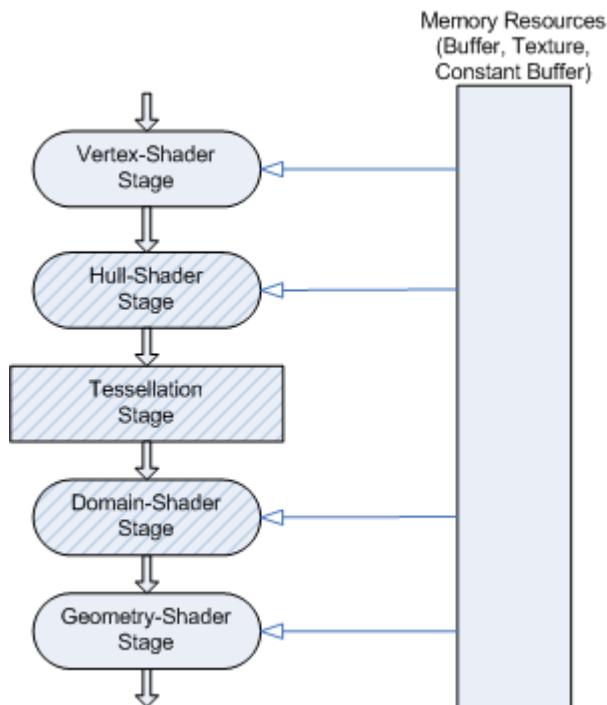
skinning/deformation models. To access the new tessellation features, you must learn about some new pipeline stages.

## New Pipeline Stages

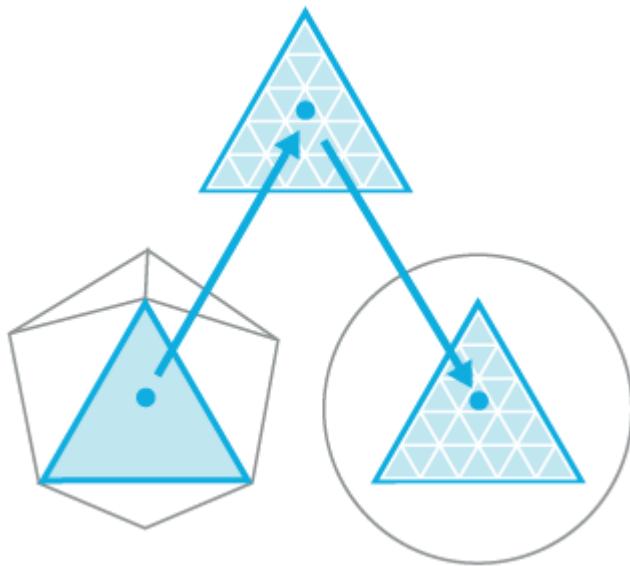
Tessellation uses the GPU to calculate a more detailed surface from a surface constructed from quad patches, triangle patches or isolines. To approximate the high-ordered surface, each patch is subdivided into triangles, points, or lines using tessellation factors. The Direct3D 11 pipeline implements tessellation using three new pipeline stages:

- **Hull-Shader Stage** - A programmable shader stage that produces a geometry patch (and patch constants) that correspond to each input patch (quad, triangle, or line).
- **Tessellator Stage** - A fixed function pipeline stage that creates a sampling pattern of the domain that represents the geometry patch and generates a set of smaller objects (triangles, points, or lines) that connect these samples.
- **Domain-Shader Stage** - A programmable shader stage that calculates the vertex position that corresponds to each domain sample.

The following diagram highlights the new stages of the Direct3D 11 pipeline.



The following diagram shows the progression through the tessellation stages. The progression starts with the low-detail subdivision surface. The progression next highlights the input patch with the corresponding geometry patch, domain samples, and triangles that connect these samples. The progression finally highlights the vertices that correspond to these samples.



## Hull-Shader Stage

A hull shader -- which is invoked once per patch -- transforms input control points that define a low-order surface into control points that make up a patch. It also does some per patch calculations to provide data for the tessellation stage and the domain stage. At the simplest black-box level, the hull-shader stage would look something like the following diagram.



A hull shader is implemented with an HLSL function, and has the following properties:

- The shader input is between 1 and 32 control points.
- The shader output is between 1 and 32 control points, regardless of the number of tessellation factors. The control-points output from a hull shader can be consumed by the domain-shader stage. Patch constant data can be consumed by a domain shader; tessellation factors can be consumed by the domain shader and the tessellation stage.
- Tessellation factors determine how much to subdivide each patch.
- The shader declares the state required by the tessellator stage. This includes information such as the number of control points, the type of patch face and the type of partitioning to use when tessellating. This information appears as declarations typically at the front of the shader code.

- If the hull-shader stage sets any edge tessellation factor to = 0 or NaN, the patch will be culled. As a result, the tessellator stage may or may not run, the domain shader will not run, and no visible output will be produced for that patch.

At a deeper level, a hull-shader actually operates in two phases: a control-point phase and a patch-constant phase, which are run in parallel by the hardware. The HLSL compiler extracts the parallelism in a hull shader and encodes it into bytecode that drives the hardware.

- The control-point phase operates once for each control-point, reading the control points for a patch, and generating one output control point (identified by a ControlPointID).
- The patch-constant phase operates once per patch to generate edge tessellation factors and other per-patch constants. Internally, many patch-constant phases may run at the same time. The patch-constant phase has read-only access to all input and output control points.

Here's an example of a hull shader:

```
[patchsize(12)]
[patchconstantfunc(MyPatchConstantFunc)]
MyOutPoint main(uint Id : SV_ControlPointID,
    InputPatch<MyInPoint, 12> InPts)
{
    MyOutPoint result;

    ...

    result = TransformControlPoint( InPts[Id] );

    return result;
}
```

For an example that creates a hull shader, see [How To: Create a Hull Shader](#).

## Tessellator Stage

The tessellator is a fixed-function stage initialized by binding a hull shader to the pipeline (see [How To: Initialize the Tessellator Stage](#)). The purpose of the tessellator stage is to subdivide a domain (quad, tri, or line) into many smaller objects (triangles, points or lines). The tessellator tiles a canonical domain in a normalized (zero-to-one) coordinate system. For example, a quad domain is tessellated to a unit square.

The tessellator operates once per patch using the tessellation factors (which specify how finely the domain will be tessellated) and the type of partitioning (which specifies the algorithm used to slice up a patch) that are passed in from the hull-shader stage. The tessellator outputs uv (and optionally w) coordinates and the surface topology to the domain-shader stage.

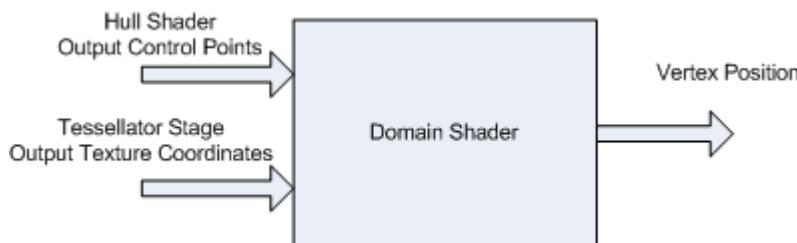
Internally, the tessellator operates in two phases:

- The first phase processes the tessellation factors, fixing rounding problems, handling very small factors, reducing and combining factors, using 32-bit floating-point arithmetic.
- The second phase generates point or topology lists based on the type of partitioning selected. This is the core task of the tessellator stage and uses 16-bit fractions with fixed-point arithmetic. Fixed-point arithmetic allows hardware acceleration while maintaining acceptable precision. For example, given a 64 meter wide patch, this precision can place points at a 2 mm resolution.

Type of Partitioning	Range
fractional_odd	[1...63]
fractional_even	TessFactor range: [2..64]
integer	TessFactor range: [1..64]
pow2	TessFactor range: [1..64]

## Domain-Shader Stage

A domain shader calculates the vertex position of a subdivided point in the output patch. A domain shader is run once per tessellator stage output point and has read-only access to the tessellator stage output UV coordinates, the hull shader output patch, and the hull shader output patch constants, as the following diagram shows.



Properties of the domain shader include:

- A domain shader is invoked once per output coordinate from the tessellator stage.
- A domain shader consumes output control points from the hull-shader stage.

- A domain shader outputs the position of a vertex.
- Inputs are the hull shader outputs including control points, patch constant data and tessellation factors. The tessellation factors can include the values used by the fixed function tessellator as well as the raw values (before rounding by integer tessellation, for example), which facilitates geomorphing, for example.

After the domain shader completes, tessellation is finished and pipeline data continues to the next pipeline stage (geometry shader, pixel shader etc). A geometry shader that expects primitives with adjacency (for example, 6 vertices per triangle) is not valid when tessellation is active (this results in undefined behavior, which the debug layer will complain about).

Here is an example of a domain shader:

```
void main( out MyDSOutput result,
           float2 myInputUV : SV_DomainPoint,
           MyDSInput DSInputs,
           OutputPatch<MyOutPoint, 12> ControlPts,
           MyTessFactors tessFactors)
{
    ...
    result.Position = EvaluateSurfaceUV(ControlPoints, myInputUV);
}
```

## APIs for initializing Tessellation Stages

Tessellation is implemented with two new programmable shader stages: a hull shader and a domain shader. These new shader stages are programmed with HLSL code that is defined in shader model 5. The new shader targets are: hs\_5\_0 and ds\_5\_0. Like all programmable shader stages, code for the hardware is extracted from compiled shaders passed into the runtime when shaders are bound to the pipeline using APIs such as [DSSetShader](#) and [HSSetShader](#). But first, the shader must be created using APIs such as [CreateHullShader](#) and [CreateDomainShader](#).

Enable tessellation by creating a hull shader and binding it to the hull-shader stage (this automatically sets up the tessellator stage). To generate the final vertex positions from the tessellated patches, you will also need to create a domain shader and bind it to the domain-shader stage. Once tessellation is enabled, the data input to the input-assembler stage must be patch data. That is, the input assembler topology must be a patch constant topology from [D3D11\\_PRIMITIVE\\_TOPOLOGY](#) set with [IASetPrimitiveTopology](#).

To disable tessellation, set the hull shader and the domain shader to **NULL**. Neither the geometry-shader stage nor the stream-output stage can read hull-shader output-control points or patch data.

- New topologies for the input-assembler stage, which are extensions to this enumeration.

```
enum D3D11_PRIMITIVE_TOPOLOGY
```

The topology is set to the input-assembler stage using [IASetPrimitiveTopology](#)

- Of course, the new programmable shader stages require other state to be set, to bind constant buffers, samples, and shader resources to the appropriate pipeline stages. These new ID3D11Device methods are implemented for setting this state.
  - [DSGetConstantBuffers](#)
  - [DSGetSamplers](#)
  - [DSGetShader](#)
  - [DSGetShaderResources](#)
  - [DSSetConstantBuffers](#)
  - [DSSetSamplers](#)
  - [DSSetShader](#)
  - [DSSetShaderResources](#)
  - [HSGetConstantBuffers](#)
  - [HSGetShaderResources](#)
  - [HSGetSamplers](#)
  - [HSGetShader](#)
  - [HSSetConstantBuffers](#)
  - [HSSetSamplers](#)
  - [HSSetShader](#)
  - [HSSetShaderResources](#)

## How to's:

The documentation also contains examples for initializing the tessellation stages.

Item	Description
<a href="#">How To: Create a Hull Shader</a>	Create a hull shader.
<a href="#">How To: Design a Hull Shader</a>	Design a hull shader.

Item	Description
<a href="#">How To: Initialize the Tessellator Stage</a>	Initialize the tessellation stage.
<a href="#">How To: Create a Domain Shader</a>	Create a domain shader.
<a href="#">How To: Design a Domain Shader</a>	Create a domain shader.

## Related topics

[Graphics Pipeline](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

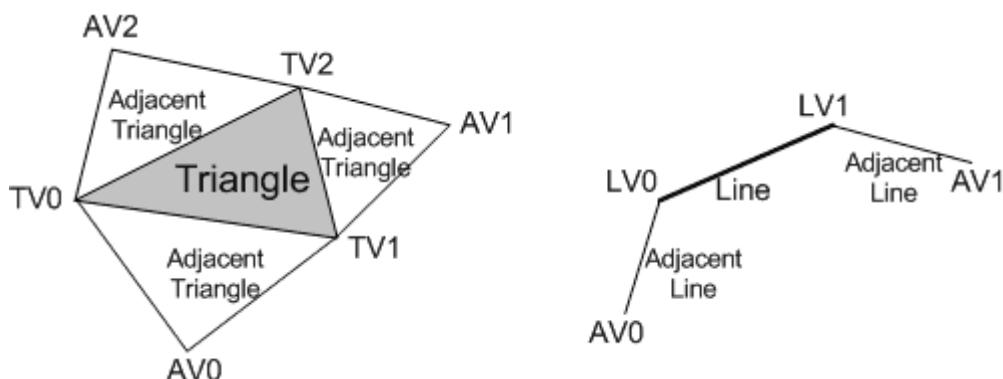
# Geometry Shader Stage

Article • 05/24/2021

The geometry-shader (GS) stage runs application-specified shader code with vertices as input and the ability to generate vertices on output.

## The Geometry Shader

Unlike vertex shaders, which operate on a single vertex, the geometry shader's inputs are the vertices for a full primitive (two vertices for lines, three vertices for triangles, or single vertex for point). Geometry shaders can also bring in the vertex data for the edge-adjacent primitives as input (an additional two vertices for a line, an additional three for a triangle). The following illustration shows a triangle and a line with adjacent vertices.



Type	
<b>TV</b>	Triangle vertex
<b>AV</b>	Adjacent vertex
<b>LV</b>	Line vertex

The geometry-shader stage can consume the `SV_PrimitiveID` [system-generated value](#) that is auto-generated by the IA. This allows per-primitive data to be fetched or computed if desired.

The geometry-shader stage is capable of outputting multiple vertices forming a single selected topology (GS stage output topologies available are: tristrip, linestrip, and pointlist). The number of primitives emitted can vary freely within any invocation of the geometry shader, though the maximum number of vertices that could be emitted must

be declared statically. Strip lengths emitted from a geometry shader invocation can be arbitrary, and new strips can be created via the [RestartStrip](#) HLSL function.

Geometry shader output may be fed to the rasterizer stage and/or to a vertex buffer in memory via the stream output stage. Output fed to memory is expanded to individual point/line/triangle lists (exactly as they would be passed to the rasterizer).

When a geometry shader is active, it is invoked once for every primitive passed down or generated earlier in the pipeline. Each invocation of the geometry shader sees as input the data for the invoking primitive, whether that is a single point, a single line, or a single triangle. A triangle strip from earlier in the pipeline would result in an invocation of the geometry shader for each individual triangle in the strip (as if the strip were expanded out into a triangle list). All the input data for each vertex in the individual primitive is available (i.e. 3 vertices for triangle), plus adjacent vertex data if applicable/available.

A geometry shader outputs data one vertex at a time by appending vertices to an output stream object. The topology of the streams is determined by a fixed declaration, choosing one of: PointStream, LineStream, or TriangleStream as the output for the GS stage. There are three types of stream objects available, PointStream, LineStream and TriangleStream which are all templated objects. The topology of the output is determined by their respective object type, while the format of the vertices appended to the stream is determined by the template type. Execution of a geometry shader instance is atomic from other invocations, except that data added to the streams is serial. The outputs of a given invocation of a geometry shader are independent of other invocations (though ordering is respected). A geometry shader generating triangle strips will start a new strip on every invocation.

When a geometry shader output is identified as a System Interpreted Value (e.g. SV\_RenderTargetArrayIndex or SV\_Position), hardware looks at this data and performs some behavior dependent on the value, in addition to being able to pass the data itself to the next shader stage for input. When such data output from the geometry shader has meaning to the hardware on a per-primitive basis (such as SV\_RenderTargetArrayIndex or SV\_ViewportArrayIndex), rather than on a per-vertex basis (such as SV\_ClipDistance[n] or SV\_Position), the per-primitive data is taken from the leading vertex emitted for the primitive.

Partially completed primitives could be generated by the geometry shader if the geometry shader ends and the primitive is incomplete. Incomplete primitives are silently discarded. This is similar to the way the IA treats partially completed primitives.

The geometry shader can perform load and texture sampling operations where screen-space derivatives are not required (samplelevel, samplecmplevelzero, samplegrad).

Algorithms that can be implemented in the geometry shader include:

- Point Sprite Expansion
- Dynamic Particle Systems
- Fur/Fin Generation
- Shadow Volume Generation
- Single Pass Render-to-Cubemap
- Per-Primitive Material Swapping
- Per-Primitive Material Setup - Including generation of barycentric coordinates as primitive data so that a pixel shader can perform custom attribute interpolation (for an example of higher-order normal interpolation, see [CubeMapGS Sample ↗](#)).

## Related topics

[Graphics Pipeline](#)

[Pipeline Stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

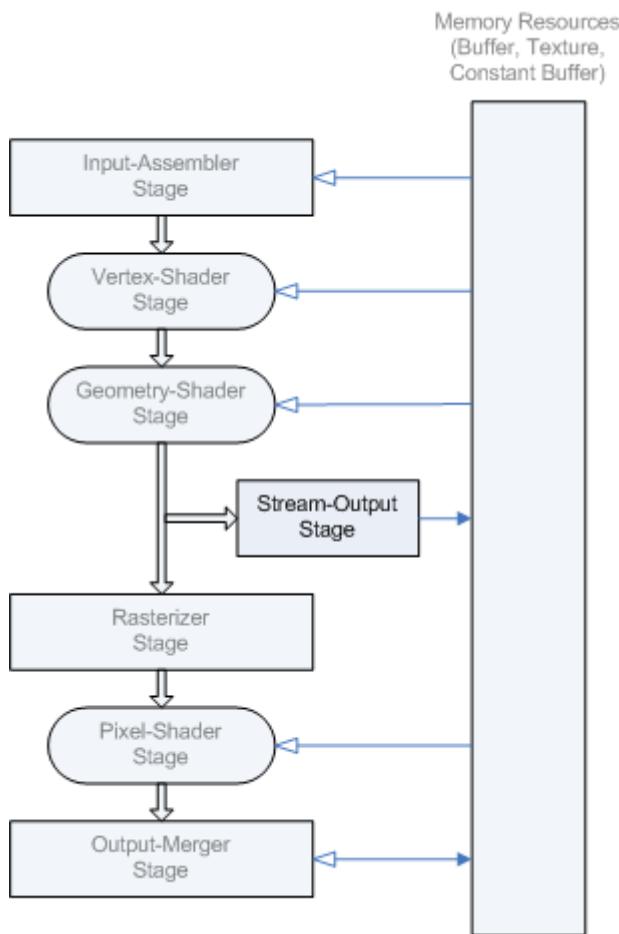
[Get help at Microsoft Q&A](#)

# Stream-Output Stage

Article • 11/04/2020

The purpose of the stream-output stage is to continuously output (or stream) vertex data from the geometry-shader stage (or the vertex-shader stage if the geometry-shader stage is inactive) to one or more buffers in memory (see [Getting Started with the Stream-Output Stage](#)).

The stream-output stage (SO) is located in the pipeline right after the geometry-shader stage and just before the rasterization stage, as shown in the following diagram.



Data streamed out to memory can be read back into the pipeline in a subsequent rendering pass, or can be copied to a staging resource (so it can be read by the CPU). The amount of data streamed out can vary; the [ID3D11DeviceContext::DrawAuto](#) API is designed to handle the data without the need to query (the GPU) about the amount of data written.

When a triangle or line strip is bound to the input-assembler stage, each strip is converted into a list before they are streamed out. Vertices are always written out as complete primitives (for example, 3 vertices at a time for triangles); incomplete primitives are never streamed out. Primitive types with adjacency discard the adjacency data before streaming data out.

There are two ways to feed stream-output data into the pipeline:

- Stream-output data can be fed back into the input-assembler stage.
- Stream-output data can be read by programmable shaders using load functions (such as [Load](#)).

To use a buffer as a stream-output resource, create the buffer with the [D3D11\\_BIND\\_STREAM\\_OUTPUT](#) flag. The stream-output stage supports up to 4 buffers simultaneously.

- If you are streaming data into multiple buffers, each buffer can only capture a single element (up to 4 components) of per-vertex data, with an implied data stride equal to the element width in each buffer (compatible with the way single element buffers can be bound for input into shader stages). Furthermore, if the buffers have different sizes, writing stops as soon as any one of the buffers is full.
- If you are streaming data into a single buffer, the buffer can capture up to 64 scalar components of per-vertex data (256 bytes or less) or the vertex stride can be up to 2048 bytes.

## In this section

Topic	Description
<a href="#">Getting Started with the Stream-Output Stage</a>	This section describes how to use a geometry shader with the stream output stage.

## Related topics

[Graphics Pipeline](#)

[Pipeline Stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Getting Started with the Stream-Output Stage

Article • 01/25/2023

This section describes how to use a geometry shader with the stream output stage.

## Compile a Geometry Shader

This geometry shader (GS) calculates a face normal for each triangle, and outputs position, normal and texture coordinate data.

```
struct GSPS_INPUT
{
    float4 Pos : SV_POSITION;
    float3 Norm : TEXCOORD0;
    float2 Tex : TEXCOORD1;
};

[maxvertexcount(12)]
void GS( triangle GSPS_INPUT input[3], inout TriangleStream<GSPS_INPUT>
TriStream )
{
    GSPS_INPUT output;

    //
    // Calculate the face normal
    //
    float3 faceEdgeA = input[1].Pos - input[0].Pos;
    float3 faceEdgeB = input[2].Pos - input[0].Pos;
    float3 faceNormal = normalize( cross(faceEdgeA, faceEdgeB) );
    float3 ExplodeAmt = faceNormal*Explode;

    //
    // Calculate the face center
    //
    float3 centerPos = (input[0].Pos.xyz + input[1].Pos.xyz +
input[2].Pos.xyz)/3.0;
    float2 centerTex = (input[0].Tex + input[1].Tex + input[2].Tex)/3.0;
    centerPos += faceNormal*Explode;

    //
    // Output the pyramid
    //
    for( int i=0; i<3; i++ )
    {
        output.Pos = input[i].Pos + float4(ExplodeAmt,0);
    }
}
```

```

        output.Pos = mul( output.Pos, View );
        output.Pos = mul( output.Pos, Projection );
        output.Norm = input[i].Norm;
        output.Tex = input[i].Tex;
        TriStream.Append( output );

        int iNext = (i+1)%3;
        output.Pos = input[iNext].Pos + float4(ExplodeAmt,0);
        output.Pos = mul( output.Pos, View );
        output.Pos = mul( output.Pos, Projection );
        output.Norm = input[iNext].Norm;
        output.Tex = input[iNext].Tex;
        TriStream.Append( output );

        output.Pos = float4(centerPos,1) + float4(ExplodeAmt,0);
        output.Pos = mul( output.Pos, View );
        output.Pos = mul( output.Pos, Projection );
        output.Norm = faceNormal;
        output.Tex = centerTex;
        TriStream.Append( output );

        TriStream.RestartStrip();
    }

    for( int i=2; i>=0; i-- )
    {
        output.Pos = input[i].Pos + float4(ExplodeAmt,0);
        output.Pos = mul( output.Pos, View );
        output.Pos = mul( output.Pos, Projection );
        output.Norm = -input[i].Norm;
        output.Tex = input[i].Tex;
        TriStream.Append( output );
    }
    TriStream.RestartStrip();
}

```

Keeping that code in mind, consider that a geometry shader looks much like a vertex or pixel shader, but with the following exceptions: the type returned by the function, the input parameter declarations, and the intrinsic function.

<b>Item</b>	<b>Description</b>
-------------	--------------------

Item	Description
Function return type	<p>The function return type does one thing, declares the maximum number of vertices that can be output by the shader. In this case,</p> <div data-bbox="382 332 1382 518" style="background-color: #f0f0f0; padding: 10px; margin: 10px auto; width: fit-content;"> <pre>maxvertexcount(12)</pre> </div> <p>defines the output to be a maximum of 12 vertices.</p>
Input parameter declarations	<p>This function takes two input parameters:</p> <div data-bbox="382 736 1382 968" style="background-color: #f0f0f0; padding: 10px; margin: 10px auto; width: fit-content;"> <pre>triangle GSPS_INPUT input[3] , inout TriangleStream&lt;GSPS_INPUTT&gt; TriStream</pre> </div>
	<p>The first parameter is an array of vertices (3 in this case) defined by a GSPS_INPUT structure (which defines per-vertex data as a position, a normal and a texture coordinate). The first parameter also uses the triangle keyword, which means the input assembler stage must output data to the geometry shader as one of the triangle primitive types (triangle list or triangle strip).</p> <p>The second parameter is a triangle stream defined by the type TriangleStream&lt;GSPS_INPUTT&gt;. This means the parameter is an array of triangles, each of which is made up of three vertices (that contain the data from the members of GSPS_INPUT).</p> <p>Use the triangle and trianglestream keywords to identify individual triangles or a stream of triangles in a GS.</p>
Intrinsic function	<p>The lines of code in the shader function use common-shader-core HLSL intrinsic functions except the last two lines, which call Append and RestartStrip. These functions are only available to a geometry shader. Append informs the geometry shader to append the output to the current strip; RestartStrip creates a new primitive strip. A new strip is implicitly created in every invocation of the GS stage.</p>

The rest of the shader looks very similar to a vertex or pixel shader. The geometry shader uses a structure to declare input parameters and marks the position member with the SV\_POSITION semantic to tell the hardware that this is positional data. The input structure identifies the other two input parameters as texture coordinates (even though one of them will contain a face normal). You could use your own custom semantic for the face normal if you prefer.

Having designed the geometry shader, call [D3DCompile](#) to compile as shown in the following code example.

```
DWORD dwShaderFlags = D3DCOMPILE_ENABLE_STRICTNESS;
ID3DBlob** ppShader;

D3DCompile( pSrcData, sizeof( pSrcData ),
    "Tutorial13.fx", NULL, NULL, "GS", "gs_4_0",
    dwShaderFlags, 0, &ppShader, NULL );
```

Just like vertex and pixel shaders, you need a shader flag to tell the compiler how you want the shader compiled (for debugging, optimized for speed, and so on), the entry point function, and the shader model to validate against. This example creates a geometry shader built from the Tutorial13.fx file, by using the GS function. The shader is compiled for shader model 4.0.

## Create a Geometry-Shader Object with Stream Output

Once you know that you will be streaming the data from the geometry, and you have successfully compiled the shader, the next step is to call

[ID3D11Device::CreateGeometryShaderWithStreamOutput](#) to create the geometry shader object.

But first, you need to declare the stream output (SO) stage input signature. This signature matches or validates the GS outputs and the SO inputs at the time of object creation. The following code is an example of the SO declaration.

```
D3D11_SO_DECLARATION_ENTRY pDecl[] =
{
    // semantic name, semantic index, start component, component count,
    output slot
    { "SV_POSITION", 0, 0, 4, 0 },    // output all components of position
    { "TEXCOORD0", 0, 0, 3, 0 },      // output the first 3 of the normal
    { "TEXCOORD1", 0, 0, 2, 0 },      // output the first 2 texture
    coordinates
};

D3D11Device->CreateGeometryShaderWithStreamOut( pShaderBytecode,
    ShaderBytecodesize, pDecl,
    sizeof(pDecl), NULL, 0, 0, NULL, &pStreamOutGS );
```

This function takes several parameters including:

- A pointer to the compiled geometry shader (or vertex shader if no geometry shader will be present and data will be streamed out directly from the vertex shader). For information about how to get this pointer, see [Getting a Pointer to a Compiled Shader](#).
- A pointer to an array of declarations that describe the input data for the stream output stage. (See [D3D11\\_SO\\_DECLARATION\\_ENTRY](#).) You can supply up to 64 declarations, one for each different type of element to be output from the SO stage. The array of declaration entries describes the data layout regardless of whether only a single buffer or multiple buffers are to be bound for stream output.
- The number of elements that are written out by the SO stage.
- A pointer to the geometry shader object that is created (see [ID3D11GeometryShader](#)).

In this situation, the buffer stride is NULL, the index of the stream to be sent to the rasterizer is 0, and the class linkage interface is NULL.

The stream output declaration defines the way that data is written to a buffer resource. You can add as many components as you want to the output declaration. Use the SO stage to write to a single buffer resource or many buffer resources. For a single buffer, the SO stage can write many different elements per-vertex. For multiple buffers, the SO stage can only write a single element of per-vertex data to each buffer.

To use the SO stage without using a geometry shader, call [ID3D11Device::CreateGeometryShaderWithStreamOutput](#) and pass a pointer to a vertex shader to the *pShaderBytecode* parameter.

## Set the Output Targets

The last step is to set the SO stage buffers. Data can be streamed into one or more buffers in memory for use later. The following code shows how to create a single buffer that can be used for vertex data, as well as for the SO stage to stream data into.

```
ID3D11Buffer *m_pBuffer;
int m_nBufferSize = 1000000;

D3D11_BUFFER_DESC bufferDesc =
{
    m_nBufferSize,
    D3D11_USAGE_DEFAULT,
    D3D11_BIND_STREAM_OUTPUT,
```

```
    0,  
    0,  
    0  
};  
D3D11Device->CreateBuffer( &bufferDesc, NULL, &m_pBuffer );
```

Create a buffer by calling [ID3D11Device::CreateBuffer](#). This example illustrates default usage, which is typical for a buffer resource that is expected to be updated fairly frequently by the CPU. The binding flag identifies the pipeline stage that the resource can be bound to. Any resource used by the SO stage must also be created with the bind flag D3D10\_BIND\_STREAM\_OUTPUT.

Once the buffer is successfully created, set it to the current device by calling [ID3D11DeviceContext::SOSetTargets](#):

```
UINT offset[1] = 0;  
D3D11Device->SOSetTargets( 1, &m_pBuffer, offset );
```

This call takes the number of buffers, a pointer to the buffers, and an array of offsets (one offset into each of the buffers that indicates where to begin writing data). Data will be written to these streaming-output buffers when a draw function is called. An internal variable keeps track of the position for where to begin writing data to the streaming-output buffers, and that variable will continue to increment until [SOSetTargets](#) is called again and a new offset value is specified.

All data written out to the target buffers will be 32-bit values.

## Related topics

[Stream-Output Stage](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Rasterizer Stage

Article • 11/04/2020

The rasterization stage converts vector information (composed of shapes or primitives) into a raster image (composed of pixels) for the purpose of displaying real-time 3D graphics.

During rasterization, each primitive is converted into pixels, while interpolating per-vertex values across each primitive. Rasterization includes clipping vertices to the view frustum, performing a divide by z to provide perspective, mapping primitives to a 2D viewport, and determining how to invoke the pixel shader. While using a pixel shader is optional, the rasterizer stage always performs clipping, a perspective divide to transform the points into homogeneous space, and maps the vertices to the viewport.

Vertices (x,y,z,w), coming into the rasterizer stage are assumed to be in homogeneous clip-space. In this coordinate space the X axis points right, Y points up and Z points away from camera.

You may disable rasterization by telling the pipeline there is no pixel shader (set the pixel shader stage to **NULL** with [ID3D11DeviceContext::PSSetShader](#)), and disabling depth and stencil testing (set **DepthEnable** and **StencilEnable** to **FALSE** in [D3D11\\_DEPTH\\_STENCIL\\_DESC](#)). While disabled, rasterization-related pipeline counters will not update. There is also a complete description of the [rasterization rules](#).

On hardware that implements hierarchical Z-buffer optimizations, you may enable preloading the z-buffer by setting the pixel shader stage to **NULL** while enabling depth and stencil testing.

## In this section

Topic	Description
<a href="#">Getting Started with the Rasterizer Stage</a>	This section describes setting the viewport, the scissors rectangle, the rasterizer state, and multi-sampling.
<a href="#">Rasterization Rules</a>	Rasterization rules define how vector data is mapped into raster data. The raster data is snapped to integer locations that are then culled and clipped (to draw the minimum number of pixels), and per-pixel attributes are interpolated (from per-vertex attributes) before being passed to a pixel shader.

# Related topics

[Graphics Pipeline](#)

[Pipeline Stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Getting Started with the Rasterizer Stage

Article • 02/28/2022

This section describes setting the viewport, the scissors rectangle, the rasterizer state, and multi-sampling.

## Set the Viewport

A viewport maps vertex positions (in clip space) into render target positions. This step scales the 3D positions into 2D space. A render target is oriented with the Y axes pointing down; this requires that the Y coordinates get flipped during the viewport scale. In addition, the x and y extents (range of the x and y values) are scaled to fit the viewport size according to the following formulas:

```
X = (X + 1) * Viewport.Width * 0.5 + Viewport.TopLeftX  
Y = (1 - Y) * Viewport.Height * 0.5 + Viewport.TopLeftY  
Z = Viewport.MinDepth + Z * (Viewport.MaxDepth - Viewport.MinDepth)
```

Tutorial 1 creates a  $640 \times 480$  viewport using [D3D11\\_VIEWPORT](#) and by calling [ID3D11DeviceContext::RSSetViewports](#).

```
D3D11_VIEWPORT vp[1];  
vp[0].Width = 640.0f;  
vp[0].Height = 480.0f;  
vp[0].MinDepth = 0;  
vp[0].MaxDepth = 1;  
vp[0].TopLeftX = 0;  
vp[0].TopLeftY = 0;  
g_pd3dContext->RSSetViewports( 1, vp );
```

The viewport description specifies the size of the viewport, the range to map depth to (using *MinDepth* and *MaxDepth*), and the placement of the top left of the viewport. *MinDepth* must be less than or equal to *MaxDepth*; the range for both *MinDepth* and *MaxDepth* is between 0.0 and 1.0, inclusive. It is common for the viewport to map to a render target but it is not necessary; additionally, the viewport does not have to have the same size or position as the render target.

You can create an array of viewports, but only one can be applied to a primitive output from the geometry shader. Only one viewport can be set active at a time. The pipeline uses a default viewport (and scissor rectangle, discussed in the next section) during rasterization. The default is always the first viewport (or scissor rectangle) in the array. To perform a per-primitive selection of the viewport in the geometry shader, specify the `ViewportArrayIndex` semantic on the appropriate GS output component in the GS output signature declaration.

The maximum number of viewports (and scissor rectangles) that can be bound to the rasterizer stage at any one time is 16 (specified by `D3D11_VIEWPORT_AND_SCISSORRECT_OBJECT_COUNT_PER_PIPELINE`).

## Set the Scissor Rectangle

A scissor rectangle gives you another opportunity to reduce the number of pixels that will be sent to the output merger stage. Pixels outside of the scissor rectangle are discarded. The size of the scissor rectangle is specified in integers. Only one scissor rectangle (based on `ViewportArrayIndex` in [system value semantics](#)) can be applied to a triangle during rasterization.

To enable the scissor rectangle, use the `ScissorEnable` member (in `D3D11_RASTERIZER_DESC1`). The default scissor rectangle is an empty rectangle; that is, all rect values are 0. In other words, if you do not set up the scissor rectangle and scissor is enabled, you will not send any pixels to the output-merger stage. The most common setup is to initialize the scissor rectangle to the size of the viewport.

To set an array of scissor rectangles to the device, call `ID3D11DeviceContext::RSSetScissorRects` with `D3D11_RECT`.

```
D3D11_RECT rect[1];
rect[0].left = 0;
rect[0].right = 640;
rect[0].top = 0;
rect[0].bottom = 480;

g_pd3dContext->RSSetScissorRects( 1, rect );
```

This method takes two parameters: (1) the number of rectangles in the array and (2) an array of rectangles.

The pipeline uses a default scissor rectangle index during rasterization (the default is a zero-size rectangle with clipping disabled). To override this, specify the

`SV_VeroprtArrayIndex` semantic to a GS output component in the GS output signature declaration. This will cause the GS stage to mark this GS output component as a system-generated component with this semantic. The rasterizer stage recognizes this semantic and will use the parameter it is attached to as the scissor rectangle index to access the array of scissor rectangles. Don't forget to tell the rasterizer stage to use the scissor rectangle that you define by enabling the `ScissorEnable` value in the rasterizer description before creating the rasterizer object.

## Set Rasterizer State

Beginning with Direct3D 10, rasterizer state is encapsulated in a rasterizer state object. You may create up to 4096 rasterizer state objects which can then be set to the device by passing a handle to the state object.

Use [ID3D11Device1::CreateRasterizerState1](#) to create a rasterizer state object from a rasterizer description (see [D3D11\\_RASTERIZER\\_DESC1](#)).

```
ID3D11RasterizerState1 * g_pRasterState;

D3D11_RASTERIZER_DESC1 rasterizerState;
rasterizerState.FillMode = D3D11_FILL_SOLID;
rasterizerState.CullMode = D3D11_CULL_FRONT;
rasterizerState.FrontCounterClockwise = true;
rasterizerState.DepthBias = false;
rasterizerState.DepthBiasClamp = 0;
rasterizerState.SlopeScaledDepthBias = 0;
rasterizerState.DepthClipEnable = true;
rasterizerState.ScissorEnable = true;
rasterizerState.MultisampleEnable = false;
rasterizerState.AntialiasedLineEnable = false;
rasterizerState.ForcedSampleCount = 0;
g_pd3dDevice->CreateRasterizerState1( &rasterizerState, &g_pRasterState
);
```

This example set of state accomplishes perhaps the most basic rasterizer setup:

- Solid fill mode
- Cull out or remove back faces; assume counter-clockwise winding order for primitives
- Turn off depth bias but enable depth buffering and enable the scissor rectangle
- Turn off multisampling and line anti-aliasing

In addition, basic rasterizer operations, always include the following: clipping (to the view frustum), perspective divide, and the viewport scale. After successfully creating the

rasterizer state object, set it to the device like this:

```
g_pd3dContext->RSSetState(g_pRasterState);
```

## Multisampling

Multisampling samples some or all of the components of an image at a higher resolution (followed by downsampling to the original resolution) to reduce the most visible form of aliasing caused by drawing polygon edges. Even though multisampling requires sub-pixel samples, modern GPU's implement multisampling so that a pixel shader runs once per pixel. This provides an acceptable tradeoff between performance (especially in a GPU bound application) and anti-aliasing the final image.

To use multisampling, set the enable field in the [rasterization description](#), create a multisampled render target, and either read the render target with a shader to resolve the samples into a single pixel color or call [ID3D11DeviceContext::ResolveSubresource](#) to resolve the samples using the video card. The most common scenario is to draw to one or more multisampled render targets.

Multisampling is independent of whether or not a sample mask is used, [alpha-to-coverage](#) is enabled, or stencil operations (which are always performed per-sample).

Depth testing is affected by multisampling:

- When multisampling is enabled, depth is interpolated per sample and the depth/stencil test is done per sample; the pixel shader output color is duplicated for all passing samples. If the pixel shader outputs depth, the depth value is duplicated for all samples (although this scenario loses the benefit of multisampling).
- When multisampling is disabled, depth/stencil testing is still done per-sample, but depth is not interpolated per-sample.

There are no restrictions for mixing multisampled and non-multisampled rendering within a single render target. If you enable multisampling and draw to a non-multisampled render target, you produce the same result as if multisampling were not enabled; sampling is done with a single sample per-pixel.

## Related topics

[Rasterizer Stage](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Rasterization Rules

Article • 08/19/2020

Rasterization rules define how vector data is mapped into raster data. The raster data is snapped to integer locations that are then culled and clipped (to draw the minimum number of pixels), and per-pixel attributes are interpolated (from per-vertex attributes) before being passed to a pixel shader.

There are several types of rules, which depend on the type of primitive that is being mapped, as well as whether or not the data uses multisampling to reduce aliasing. The following illustrations demonstrate how the corner cases are handled.

- [Triangle Rasterization Rules \(Without Multisampling\)](#)
- [Line Rasterization Rules \(Aliased, Without Multisampling\)](#)
- [Line Rasterization Rules \(Antialiased, Without Multisampling\)](#)
- [Point Rasterization Rules \(Without Multisampling\)](#)
- [Multisample Anti-Aliasing Rasterization Rules](#)
  - [Hardware Support](#)
  - [Centroid Sampling of Attributes when Multisample Antialiasing](#)
  - [Derivative Calculations When Multisampling](#)
- [Related topics](#)

## Triangle Rasterization Rules (Without Multisampling)

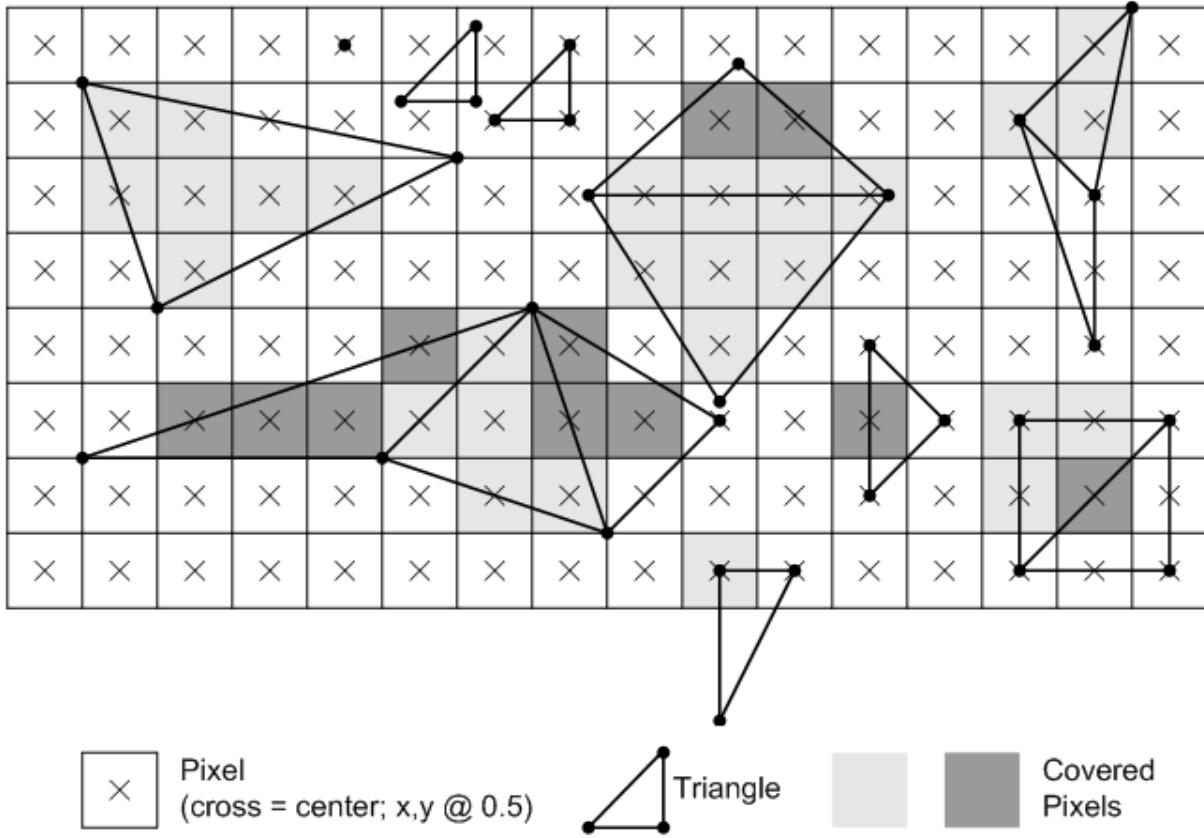
Any pixel center which falls inside a triangle is drawn; a pixel is assumed to be inside if it passes the top-left rule. The top-left rule is that a pixel center is defined to lie inside of a triangle if it lies on the top edge or the left edge of a triangle.

Where:

- A top edge, is an edge that is exactly horizontal and is above the other edges.
- A left edge, is an edge that is not exactly horizontal and is on the left side of the triangle. A triangle can have one or two left edges.

The top-left rule ensures that adjacent triangles are drawn once.

This illustration shows examples of pixels that are drawn because they either lie inside a triangle or follow the top-left rule.



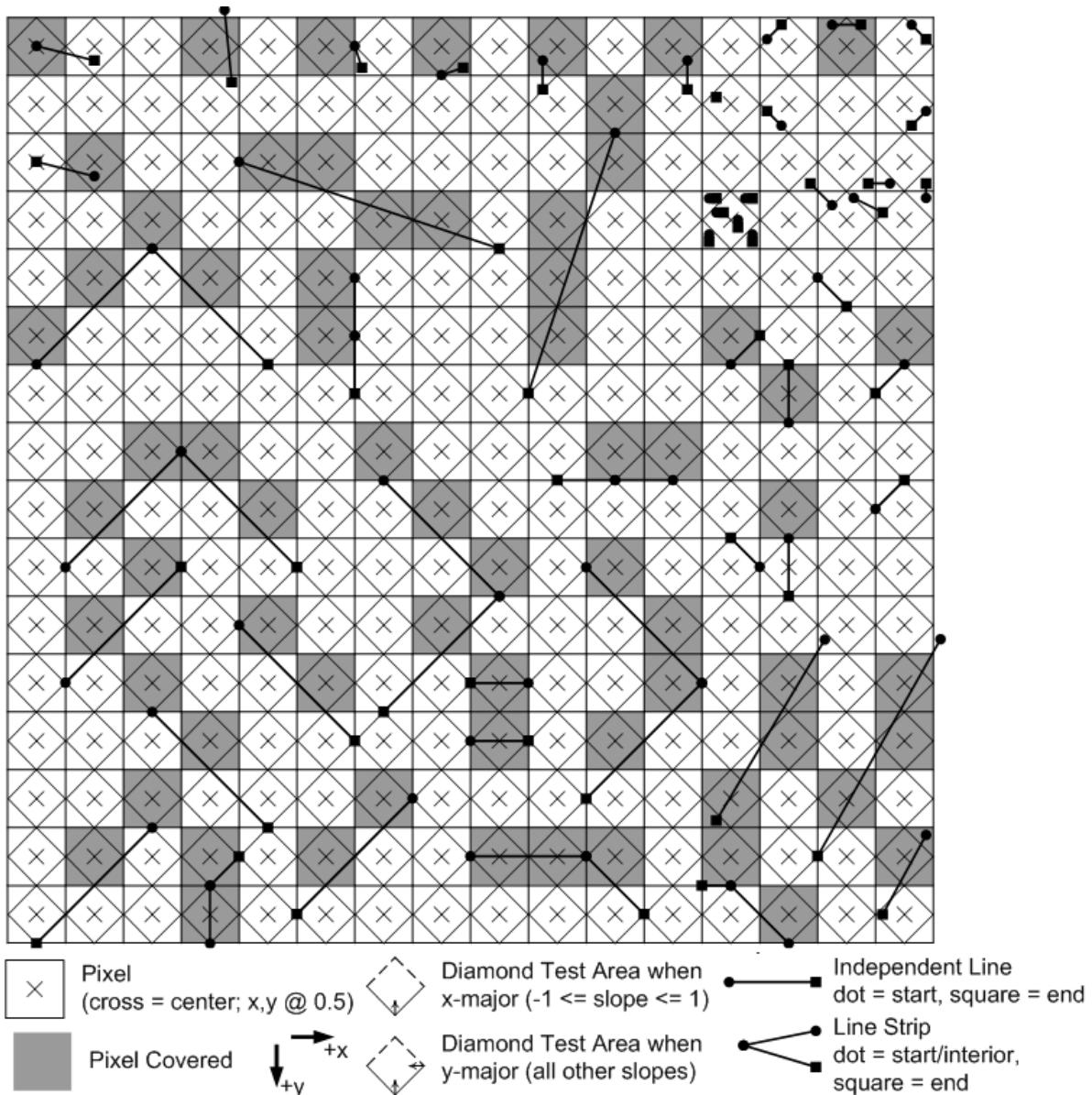
The light and dark gray covering of the pixels show them as groups of the pixels to indicate which triangle they are inside.

## Line Rasterization Rules (Aliased, Without Multisampling)

Line rasterization rules use a diamond test area to determine if a line covers a pixel. For x-major lines (lines with  $-1 \leq \text{slope} \leq +1$ ), the diamond test area includes (shown solid) the lower-left edge, lower-right edge, and bottom corner; the diamond excludes (shown dotted) the upper-left edge, upper-right edge, the top corner, the left corner, and the right corner. A y-major line is any line that is not an x-major line; the test diamond area is the same as described for the x-major line except the right corner is also included.

Given the diamond area, a line covers a pixel if the line exits the pixel's diamond test area when traveling along the line from the start towards the end. A line strip behaves the same, as it is drawn as a sequence of lines.

The following illustration shows some examples.

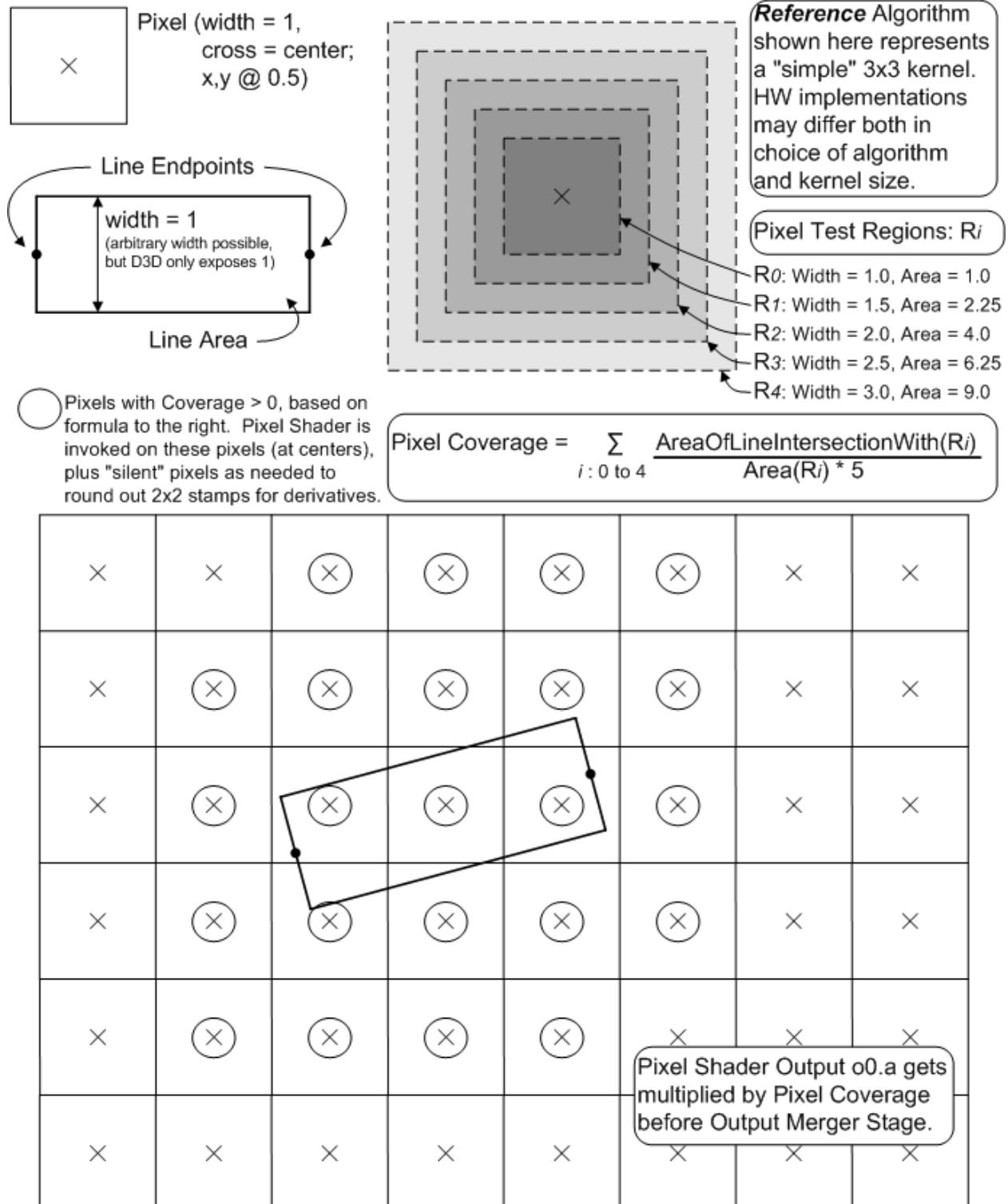


## Line Rasterization Rules (Antialiased, Without Multisampling)

An antialiased line is rasterized as if it were a rectangle (with width = 1). The rectangle intersects with a render target producing per-pixel coverage values, which are multiplied into pixel shader output alpha components. There is no antialiasing performed when drawing lines on a multisampled render target.

It is deemed that there is no single "best" way to perform antialiased line rendering. Direct3D 10 adopts as a guideline the method shown in the following illustration. This method was derived empirically, exhibiting a number of visual properties deemed desirable. Hardware need not exactly match this algorithm; tests against this reference shall have "reasonable" tolerances, guided by some of the principles listed further below, permitting various hardware implementations and filter kernel sizes. None of this

flexibility permitted in hardware implementation, however, can be communicated up through Direct3D 10 to applications, beyond simply drawing lines and observing/measuring how they look.



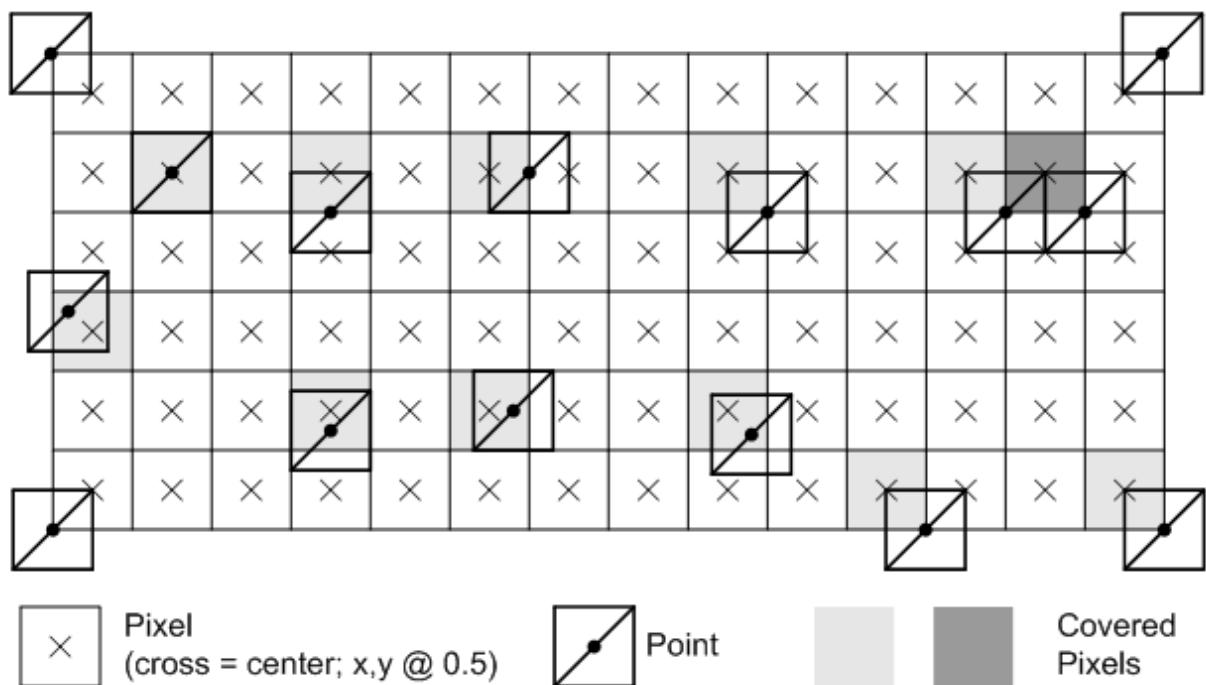
This algorithm generates relatively smooth lines, with uniform intensity, with minimal jagged edges or braiding. Moire patterning for close lines is minimized. There is good coverage for junctions between line segments placed end-to-end. The filter kernel is a reasonable tradeoff between the amount of edge blurring and the changes in intensity caused by gamma corrections. The coverage value is multiplied into pixel shader  $o0.a$ .

( $\text{srcAlpha}$ ) per the following formula by the output-merger stage:  $\text{srcColor} * \text{srcAlpha} + \text{destColor} * (1-\text{srcAlpha})$ .

## Point Rasterization Rules (Without Multisampling)

A point is interpreted as though it were composed of two triangles in a Z pattern, which use triangle rasterization rules. The coordinate identifies the center of a one pixel wide square. There is no culling for points.

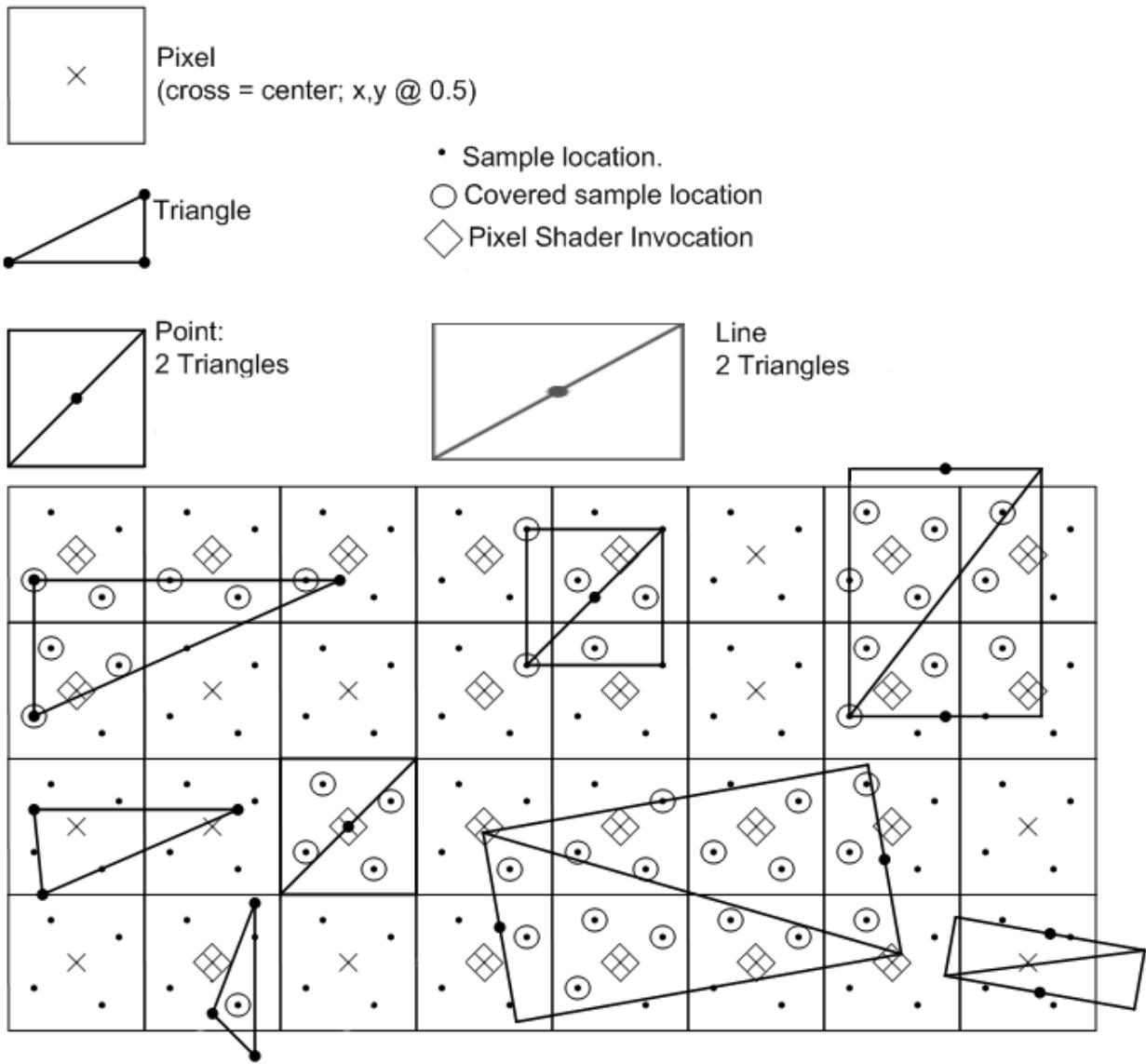
The following illustration shows some examples.



## Multisample Anti-Aliasing Rasterization Rules

Multisample antialiasing (MSAA) reduces geometry aliasing using pixel coverage and depth-stencil tests at multiple sub-sample locations. To improve performance, per-pixel calculations are performed once for each covered pixel, by sharing shader outputs across covered sub-pixels. Multisample antialiasing does not reduce surface aliasing. Sample locations and reconstruction functions are dependent on the hardware implementation.

The following illustration shows some examples.



The number of sample locations is dependent on the multisample mode. Vertex attributes are interpolated at pixel centers, since this is where the pixel shader is invoked (this becomes extrapolation if the center is not covered). Attributes can be flagged in the pixel shader to be centroid sampled, which causes non-covered pixels to interpolate the attribute at intersection of the pixel's area and the primitive. A pixel shader runs for each 2x2 pixel area to support derivative calculations (which use x and y deltas). This means that shader invocations occur more than is shown to fill out the minimum 2x2 quanta (which is independent of multisampling). The shader result is written out for each covered sample that passes the per-sample depth-stencil test.

Rasterization rules for primitives are, in general, unchanged by multisample antialiasing, except:

- For a triangle, a coverage test is performed for each sample location (not for a pixel center). If more than one sample location is covered, a pixel shader runs once with attributes interpolated at the pixel center. The result is stored (replicated) for each covered sample location in the pixel that passes the depth/stencil test.

A line is treated as a rectangle made up of two triangles, with a line width of 1.4.

- For a point, a coverage test is performed for each sample location (not for a pixel center).

Many formats support multisampling (see [Hardware Support for Direct3D 10 Formats](#)), some formats can be resolved ([ResolveSubresource](#); which downsamples a multisampled format to a sample size of 1). Multisampling formats can be used in render targets which can be read back into shaders using [load](#), since no resolve is required for individual samples accessed by the shader. Depth formats are not supported for multisample resource, therefore, depth formats are restricted to render targets only.

Typeless formats (R8G8B8A8\_TYPELESS for instance) support multisampling to allow a resource view to interpret data in different ways. For instance, you could create a multisample resource using R8G8B8A8\_TYPELESS, render to it using a render-target-view resource with a R8G8B8A8\_UINT format, then resolve the contents to another resource with a R8G8B8A8\_UNORM data format.

## Hardware Support

The API reports hardware support for multisampling through the number of quality levels. For example, a 0 quality level means the hardware does not support multisampling (at a particular format and quality level). A 3 for quality levels means that the hardware supports three different sample layouts and/or resolve algorithms. You can also assume the following:

- Any format that supports multisampling, supports the same number of quality levels for every format in that family.
- Every format that supports multisampling, and has the \_UNORM, \_SRGB, \_SNORM or \_FLOAT formats, also supports resolving.

## Centroid Sampling of Attributes when Multisample Antialiasing

By default, vertex attributes are interpolated to a pixel center during multisample antialiasing; if the pixel center is not covered, attributes are extrapolated to a pixel center. If a pixel shader input that contains the centroid semantic (assuming the pixel is not fully covered) will be sampled somewhere within the covered area of the pixel, possibly at one of the covered sample locations. A sample mask (specified by the rasterizer state) is applied prior to centroid computation. Therefore, a sample that is masked out will not be used as a centroid location.

The reference rasterizer chooses a sample location for centroid sampling similar to this:

- The sample mask allows all samples. Use a pixel center if the pixel is covered or if none of the samples are covered. Otherwise, the first covered sample is chosen, starting from the pixel center and moving outward.
- The sample mask turns off all samples but one (a common scenario). An application can implement multipass supersampling by cycling through single-bit sample-mask values and re-rendering the scene for each sample using centroid sampling. This would require that an application adjust derivatives to select appropriately more detailed texture mips for the higher texture sampling density.

## Derivative Calculations When Multisampling

Pixel shaders always run using a minimum 2x2 pixel area to support derivative calculations, which are calculated by taking deltas between data from adjacent pixels (making the assumption that the data in each pixel has been sampled with unit spacing horizontally or vertically). This is unaffected by multisampling.

If derivatives are requested on an attribute that has been centroid sampled, the hardware calculation is not adjusted, which can cause inaccurate derivatives. A shader will expect a unit vector in render-target space but may get a non-unit vector with respect to some other vector space. Therefore, it is an application's responsibility to exhibit caution when requesting derivatives from attributes that are centroid sampled. In fact, it is recommended that you do not combine derivatives and centroid sampling. Centroid sampling can be useful for situations where it is critical that a primitive's interpolated attributes are not extrapolated, but this comes with tradeoffs such as attributes that appear to jump where a primitive edge crosses a pixel (rather than change continuously) or derivatives that cannot be used by texture sampling operations that derive LOD.

## Related topics

[Rasterizer Stage](#)

---

## Feedback



Was this page helpful? [!\[\]\(8819a0cf8314453473a6185159129ff6\_img.jpg\) Yes](#) [!\[\]\(19ade057969243d1981c6bc618530629\_img.jpg\) No](#)

Get help at Microsoft Q&A

# Pixel Shader Stage

Article • 08/19/2020

The pixel-shader stage (PS) enables rich shading techniques such as per-pixel lighting and post-processing. A pixel shader is a program that combines constant variables, texture data, interpolated per-vertex values, and other data to produce per-pixel outputs. The rasterizer stage invokes a pixel shader once for each pixel covered by a primitive, however, it is possible to specify a **NULL** shader to avoid running a shader.

## The Pixel Shader

When multisampling a texture, a pixel shader is invoked once per-covered pixel while a depth/stencil test occurs for each covered multisample. Samples that pass the depth/stencil test are updated with the pixel shader output color.

The pixel shader intrinsic functions produce or use derivatives of quantities with respect to screen space x and y. The most common use for derivatives is to compute level-of-detail calculations for texture sampling and in the case of anisotropic filtering, selecting samples along the axis of anisotropy. Typically, a hardware implementation runs a pixel shader on multiple pixels (for example a 2x2 grid) simultaneously, so that derivatives of quantities computed in the pixel shader can be reasonably approximated as deltas of the values at the same point of execution in adjacent pixels.

## Inputs

When the pipeline is configured without a geometry shader, a pixel shader is limited to 16, 32-bit, 4-component inputs. Otherwise, a pixel shader can take up to 32, 32-bit, 4-component inputs.

Pixel shader input data includes vertex attributes (that can be interpolated with or without perspective correction) or can be treated as per-primitive constants. Pixel shader inputs are interpolated from the vertex attributes of the primitive being rasterized, based on the interpolation mode declared. If a primitive gets clipped before rasterization, the interpolation mode is honored during the clipping process as well.

Vertex attributes are interpolated (or evaluated) at pixel shader center locations. Pixel shader attribute interpolation modes are declared in an input register declaration, on a per-element basis in either an [argument](#) or an [input structure](#). Attributes can be interpolated linearly, or with [centroid sampling](#). Centroid evaluation is relevant only during multisampling to cover cases where a pixel is covered by a primitive but a pixel

center may not be; centroid evaluation occurs as close as possible to the (non-covered) pixel center.

Inputs may also be declared with a [system-value semantic](#), which marks a parameter that is consumed by other pipeline stages. For instance, a pixel position should be marked with the SV\_Position semantic. The IA stage can produce one scalar for a pixel shader (using SV\_PrimitiveID); the rasterizer stage can also generate one scalar for a pixel shader (using SV\_IsFrontFace).

## Outputs

A pixel shader can output up to 8, 32-bit, 4-component colors, or no color if the pixel is discarded. Pixel shader output register components must be declared before they can be used; each register is allowed a distinct output-write mask.

Use the depth-write-enable state (in the output-merger stage) to control whether depth data gets written to a depth buffer (or use the discard instruction to discard data for that pixel). A pixel shader can also output an optional 32-bit, 1-component, floating-point, depth value for depth testing (using the SV\_Depth semantic). The depth value is output in the oDepth register, and replaces the interpolated depth value for depth testing (assuming depth testing is enabled). There is no way to dynamically change between using fixed-function depth or shader oDepth.

A pixel shader cannot output a stencil value.

## Related topics

[Graphics Pipeline](#)

[Pipeline Stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Output-Merger Stage

Article • 05/24/2021

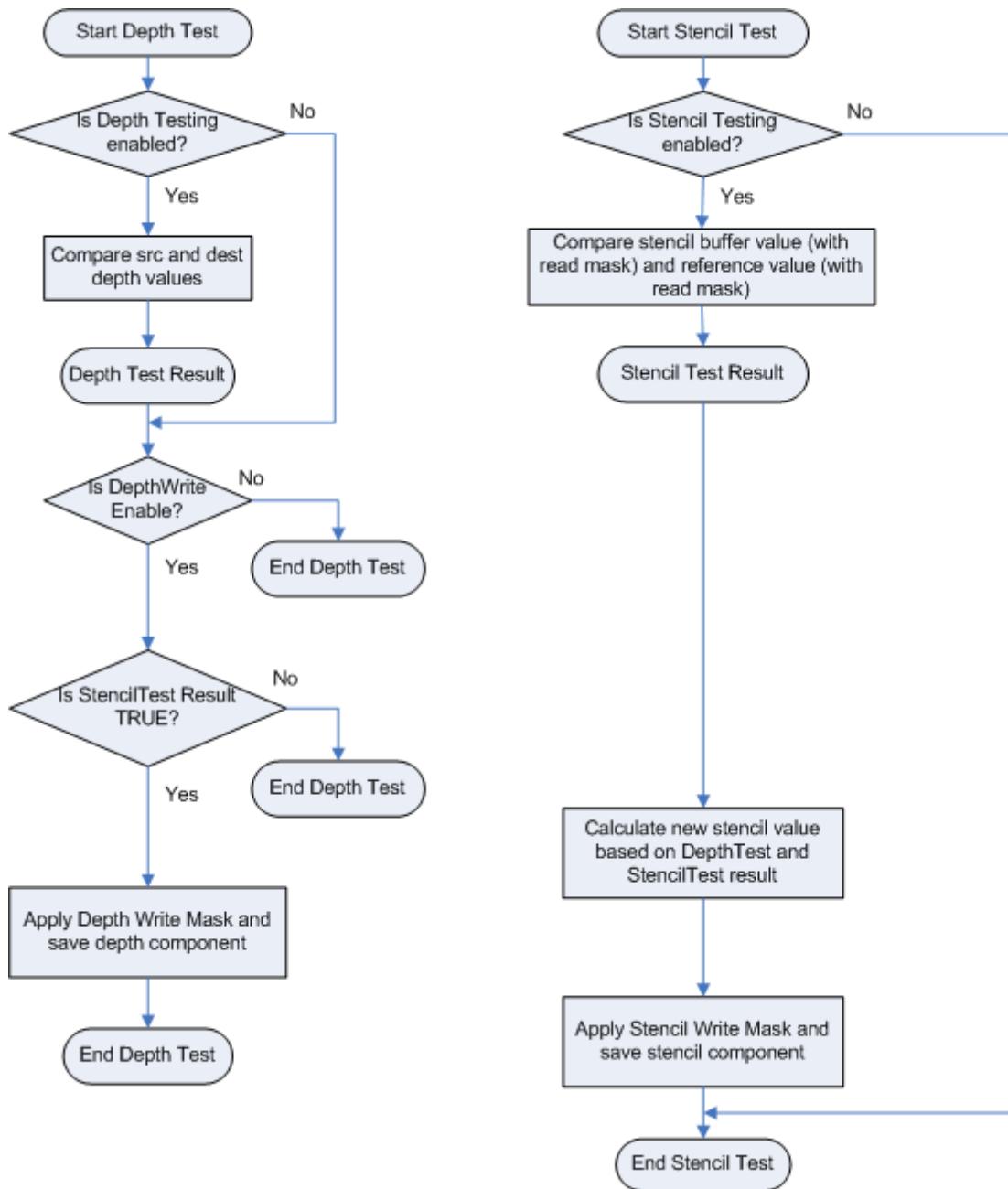
The output-merger (OM) stage generates the final rendered pixel color using a combination of pipeline state, the pixel data generated by the pixel shaders, the contents of the render targets, and the contents of the depth/stencil buffers. The OM stage is the final step for determining which pixels are visible (with depth-stencil testing) and blending the final pixel colors.

Differences between Direct3D 9 and Direct3D 10:

- Direct3D 9 implements alpha testing (using [alpha-testing state](#)) to control whether a pixel is written to an output render target.
- Direct3D 10 and higher does not implement an alpha test (or alpha testing state). This can be controlled using a pixel shader or with depth/stencil functionality.

## Depth-Stencil Testing Overview

A depth-stencil buffer, which is created as a texture resource, can contain both depth data and stencil data. The depth data is used to determine which pixels lie closest to the camera, and the stencil data is used to mask which pixels can be updated. Ultimately, both the depth and stencil values data are used by the output-merger stage to determine if a pixel should be drawn or not. The following diagram shows conceptually how depth-stencil testing is done.



To configure depth-stencil testing, see [Configuring Depth-Stencil Functionality](#). A depth-stencil object encapsulates depth-stencil state. An application can specify depth-stencil state, or the OM stage will use default values. Blending operations are performed on a per-pixel basis if multisampling is disabled. If multisampling is enabled, blending occurs on a per-multisample basis.

The process of using the depth buffer to determine which pixel should be drawn is called depth buffering, also sometimes called z-buffering.

Once depth values reach the output-merger stage (whether coming from interpolation or from a pixel shader) they are always clamped:  $z = \min(\text{Viewport.MaxDepth}, \max(\text{Viewport.MinDepth}, z))$  according to the format/precision of the depth buffer, using floating-point rules. After clamping, the depth value is compared (using DepthFunc) against the existing depth-buffer value. If no depth buffer is bound, the depth test always passes.

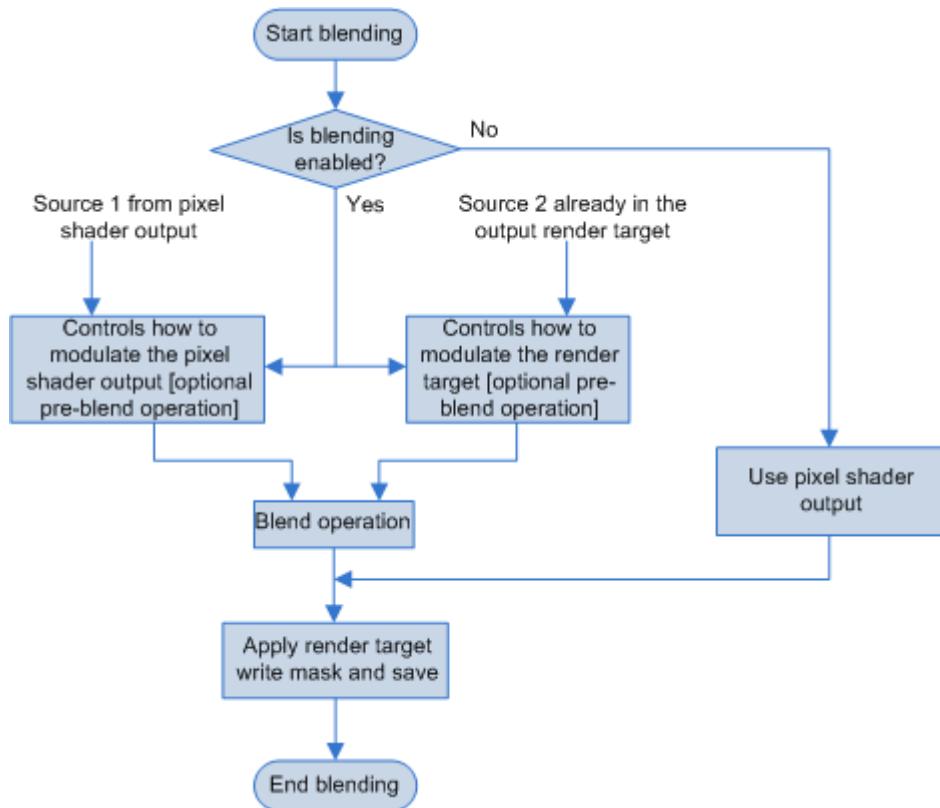
If there is no stencil component in the depth-buffer format, or no depth buffer bound, then the stencil test always passes. Otherwise, functionality is unchanged from Direct3D 9.

Only one depth/stencil buffer can be active at a time; any bound resource view must match (same size and dimensions) the depth/stencil view. This does not mean the resource size must match, just that the view size must match.

For more information about depth-stencil testing, see [tutorial 14](#).

## Blending Overview

Blending combines one or more pixel values to create a final pixel color. The following diagram shows the process involved in blending pixel data.



Conceptually, you can visualize this flow chart implemented twice in the output-merger stage: the first one blends RGB data, while in parallel, a second one blends alpha data.

To see how to use the API to create and set blend state, see [Configuring Blending Functionality](#).

Fixed-function blend can be enabled independently for each render target. However there is only one set of blend controls, so that the same blend is applied to all RenderTargets with blending enabled. Blend values (including BlendFactor) are always clamped to the range of the render-target format before blending. Clamping is done per render target, respecting the render target type. The only exception is for the

float16, float11 or float10 formats which are not clamped so that blend operations on these formats can be done with at least equal precision/range as the output format. NaN's and signed zeros are propagated for all cases (including 0.0 blend weights).

When you use sRGB render targets, the runtime converts the render target color into linear space before it performs blending. The runtime converts the final blended value back into sRGB space before it saves the value back to the render target.

Differences between Direct3D 9 and Direct3D 10:

- In Direct3D 9, fixed-function blending can be enabled independently for each render target.
- In Direct3D 10 and higher, there is one blend-state description; therefore, one blending value can be set for all render targets.

## Dual-Source Color Blending

This feature enables the output-merger stage to simultaneously use both pixel shader outputs ( $o_0$  and  $o_1$ ) as inputs to a blending operation with the single render target at slot 0. Valid blend operations include: add, subtract and revsubtract. Valid blend options for SrcBlend, DestBlend, SrcBlendAlpha or DestBlendAlpha include:

`D3D11_BLEND_SRC1_COLOR`, `D3D11_BLEND_INV_SRC1_COLOR`,

`D3D11_BLEND_SRC1_ALPHA`, `D3D11_BLEND_INV_SRC1_ALPHA`. The blend equation and the output write mask specify which components the pixel shader is outputting. Extra components are ignored.

Writing to other pixel shader outputs ( $o_2$ ,  $o_3$  etc.) is undefined; you may not write to a render target if it is not bound to slot 0. Writing  $o_{\text{Depth}}$  is valid during dual source color blending.

For examples, see [blending pixel shader outputs](#).

## Multiple RenderTargets Overview

A pixel shader can be used to render to at least 8 separate render targets, all of which must be the same type (buffer, Texture1D, Texture1DArray, and so on). Furthermore, all render targets must have the same size in all dimensions (width, height, depth, array size, sample counts). Each render target may have a different data format.

You may use any combination of render targets slots (up to 8). However, a resource view cannot be bound to multiple render-target-slots simultaneously. A view may be reused as long as the resources are not used simultaneously.

# Output-Write Mask Overview

Use an output-write mask to control (per component) what data can be written to a render target.

## Sample Mask Overview

A sample mask is a 32-bit multisample coverage mask that determines which samples get updated in active render targets. Only one sample mask is allowed. The mapping of bits in a sample mask to the samples in a resource is defined by a user. For n-sample rendering, the first n bits (from the LSB) of the sample mask are used (32 bits if the maximum number of bits).

## In this section

Topic	Description
Configuring Depth-Stencil Functionality	This section covers the steps for setting up the depth-stencil buffer, and depth-stencil state for the output-merger stage.
Configuring Blending Functionality	Blending operations are performed on every pixel shader output (RGBA value) before the output value is written to a render target. If multisampling is enabled, blending is done on each multisample; otherwise, blending is performed on each pixel.
Depth Bias	Polygons that are coplanar in 3D space can be made to appear as if they are not coplanar by adding a z-bias (or depth bias) to each one.

## Related topics

[Graphics Pipeline](#)

[Pipeline Stages \(Direct3D 10\)](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Configuring Depth-Stencil Functionality

Article • 08/19/2020

This section covers the steps for setting up the depth-stencil buffer, and depth-stencil state for the output-merger stage.

- [Create a Depth-Stencil Resource](#)
- [Create Depth-Stencil State](#)
- [Bind Depth-Stencil Data to the OM Stage](#)

Once you know how to use the depth-stencil buffer and the corresponding depth-stencil state, refer to [advanced-stencil techniques](#).

## Create a Depth-Stencil Resource

Create the depth-stencil buffer using a texture resource.

```
ID3D11Texture2D* pDepthStencil = NULL;
D3D11_TEXTURE2D_DESC descDepth;
descDepth.Width = backBufferSurfaceDesc.Width;
descDepth.Height = backBufferSurfaceDesc.Height;
descDepth.MipLevels = 1;
descDepth.ArraySize = 1;
descDepth.Format = pDeviceSettings->d3d11.AutoDepthStencilFormat;
descDepth.SampleDesc.Count = 1;
descDepth.SampleDesc.Quality = 0;
descDepth.Usage = D3D11_USAGE_DEFAULT;
descDepth.BindFlags = D3D11_BIND_DEPTH_STENCIL;
descDepth.CPUAccessFlags = 0;
descDepth.MiscFlags = 0;
hr = pd3dDevice->CreateTexture2D( &descDepth, NULL, &pDepthStencil );
```

## Create Depth-Stencil State

The depth-stencil state tells the output-merger stage how to perform the [depth-stencil test](#). The depth-stencil test determines whether or not a given pixel should be drawn.

```
D3D11_DEPTH_STENCIL_DESC dsDesc;
// Depth test parameters
```

```

dsDesc.DepthEnable = true;
dsDesc.DepthWriteMask = D3D11_DEPTH_WRITE_MASK_ALL;
dsDesc.DepthFunc = D3D11_COMPARISON_LESS;

// Stencil test parameters
dsDesc.StencilEnable = true;
dsDesc.StencilReadMask = 0xFF;
dsDesc.StencilWriteMask = 0xFF;

// Stencil operations if pixel is front-facing
dsDesc.FrontFace.StencilFailOp = D3D11_STENCIL_OP_KEEP;
dsDesc.FrontFace.StencilDepthFailOp = D3D11_STENCIL_OP_INCR;
dsDesc.FrontFace.StencilPassOp = D3D11_STENCIL_OP_KEEP;
dsDesc.FrontFace.StencilFunc = D3D11_COMPARISON_ALWAYS;

// Stencil operations if pixel is back-facing
dsDesc.BackFace.StencilFailOp = D3D11_STENCIL_OP_KEEP;
dsDesc.BackFace.StencilDepthFailOp = D3D11_STENCIL_OP_DECR;
dsDesc.BackFace.StencilPassOp = D3D11_STENCIL_OP_KEEP;
dsDesc.BackFace.StencilFunc = D3D11_COMPARISON_ALWAYS;

// Create depth stencil state
ID3D11DepthStencilState * pDSState;
pd3dDevice->CreateDepthStencilState(&dsDesc, &pDSState);

```

DepthEnable and StencilEnable enable (and disable) depth and stencil testing. Set DepthEnable to **FALSE** to disable depth testing and prevent writing to the depth buffer. Set StencilEnable to **FALSE** to disable stencil testing and prevent writing to the stencil buffer (when DepthEnable is **FALSE** and StencilEnable is **TRUE**, the depth test always passes in the stencil operation).

DepthEnable only affects the output-merger stage - it does not affect clipping, depth bias, or clamping of values before the data is input to a pixel shader.

## Bind Depth-Stencil Data to the OM Stage

Bind the depth-stencil state.

```

// Bind depth stencil state
pDevice->OMSetDepthStencilState(pDSState, 1);

```

Bind the depth-stencil resource using a view.

```

D3D11_DEPTH_STENCIL_VIEW_DESC descDSV;
descDSV.Format = DXGI_FORMAT_D32_FLOAT_S8X24_UINT;
descDSV.ViewDimension = D3D11_DSV_DIMENSION_TEXTURE2D;
descDSV.Texture2D.MipSlice = 0;

// Create the depth stencil view
ID3D11DepthStencilView* pDSV;
hr = pd3dDevice->CreateDepthStencilView( pDepthStencil, // Depth stencil
texture
                                         &descDSV, // Depth stencil desc
                                         &pDSV ); // [out] Depth stencil
view

// Bind the depth stencil view
pd3dDeviceContext->OMSetRenderTargets( 1, // One rendertarget view
                                         &pRTV, // Render target view, created
earlier
                                         pDSV ); // Depth stencil view for the
render target

```

An array of render-target views may be passed into [ID3D11DeviceContext::OMSetRenderTargets](#), however all of those render-target views will correspond to a single depth stencil view. The render target array in Direct3D 11 is a feature that enables an application to render onto multiple render targets simultaneously at the primitive level. Render target arrays offer increased performance over individually setting render targets with multiple calls to [ID3D11DeviceContext::OMSetRenderTargets](#) (essentially the method employed in Direct3D 9).

Render targets must all be the same type of resource. If multisample antialiasing is used, all bound render targets and depth buffers must have the same sample counts.

When a buffer is used as a render target, depth-stencil testing and multiple render targets are not supported.

- As many as 8 render targets can be bound simultaneously.
- All render targets must have the same size in all dimensions (width and height, and depth for 3D or array size for \*Array types).
- Each render target may have a different data format.
- Write masks control what data gets written to a render target. The output write masks control on a per-render target, per-component level what data gets written to the render target(s).

## Advanced Stencil Techniques

The stencil portion of the depth-stencil buffer can be used for creating rendering effects such as compositing, decaling, and outlining.

- Compositing
- Decaling
- Outlines and Silhouettes
- Two-Sided Stencil
- Reading the Depth-Stencil Buffer as a Texture

## Compositing

Your application can use the stencil buffer to composite 2D or 3D images onto a 3D scene. A mask in the stencil buffer is used to occlude an area of the rendering target surface. Stored 2D information, such as text or bitmaps, can then be written to the occluded area. Alternately, your application can render additional 3D primitives to the stencil-masked region of the rendering target surface. It can even render an entire scene.

Games often composite multiple 3D scenes together. For instance, driving games typically display a rear-view mirror. The mirror contains the view of the 3D scene behind the driver. It is essentially a second 3D scene composited with the driver's forward view.

## Decaling

Direct3D applications use decaling to control which pixels from a particular primitive image are drawn to the rendering target surface. Applications apply decals to the images of primitives to enable coplanar polygons to render correctly.

For instance, when applying tire marks and yellow lines to a roadway, the markings should appear directly on top of the road. However, the z values of the markings and the road are the same. Therefore, the depth buffer might not produce a clean separation between the two. Some pixels in the back primitive may be rendered on top of the front primitive and vice versa. The resulting image appears to shimmer from frame to frame. This effect is called z-fighting or flimmering.

To solve this problem, use a stencil to mask the section of the back primitive where the decal will appear. Turn off z-buffering and render the image of the front primitive into the masked-off area of the render-target surface.

Multiple texture blending can be used to solve this problem.

## Outlines and Silhouettes

You can use the stencil buffer for more abstract effects, such as outlining and silhouetting.

If your application does two render passes - one to generate the stencil mask and second to apply the stencil mask to the image, but with the primitives slightly smaller on the second pass - the resulting image will contain only the primitive's outline. The application can then fill the stencil-masked area of the image with a solid color, giving the primitive an embossed look.

If the stencil mask is the same size and shape as the primitive you are rendering, the resulting image contains a hole where the primitive should be. Your application can then fill the hole with black to produce a silhouette of the primitive.

## Two-Sided Stencil

Shadow Volumes are used for drawing shadows with the stencil buffer. The application computes the shadow volumes cast by occluding geometry, by computing the silhouette edges and extruding them away from the light into a set of 3D volumes. These volumes are then rendered twice into the stencil buffer.

The first render draws forward-facing polygons, and increments the stencil-buffer values. The second render draws the back-facing polygons of the shadow volume, and decrements the stencil buffer values. Normally, all incremented and decremented values cancel each other out. However, the scene was already rendered with normal geometry causing some pixels to fail the z-buffer test as the shadow volume is rendered. Values left in the stencil buffer correspond to pixels that are in the shadow. These remaining stencil-buffer contents are used as a mask, to alpha-blend a large, all-encompassing black quad into the scene. With the stencil buffer acting as a mask, the result is to darken pixels that are in the shadows.

This means that the shadow geometry is drawn twice per light source, hence putting pressure on the vertex throughput of the GPU. The two-sided stencil feature has been designed to mitigate this situation. In this approach, there are two sets of stencil state (named below), one set each for the front-facing triangles and the other for the back-facing triangles. This way, only a single pass is drawn per shadow volume, per light.

An example of two-sided stencil implementation can be found in the [ShadowVolume10 Sample](#).

## Reading the Depth-Stencil Buffer as a Texture

An inactive depth-stencil buffer can be read by a shader as a texture. An application that reads a depth-stencil buffer as a texture renders in two passes, the first pass writes to the depth-stencil buffer and the second pass reads from the buffer. This allows a shader to compare depth or stencil values previously written to the buffer against the value for the pixel currently being rendered. The result of the comparison can be used to create effects such as shadow mapping or soft particles in a particle system.

To create a depth-stencil buffer that can be used as both a depth-stencil resource and a shader resource a few changes need to be made to sample code in the [Create a Depth-Stencil Resource](#) section.

- The depth-stencil resource must have a typeless format such as `DXGI_FORMAT_R32_TYPELESS`.

```
descDepth.Format = DXGI_FORMAT_R32_TYPELESS;
```

- The depth-stencil resource must use both the `D3D10_BIND_DEPTH_STENCIL` and `D3D10_BIND_SHADER_RESOURCE` bind flags.

```
descDepth.BindFlags = D3D10_BIND_DEPTH_STENCIL |  
D3D10_BIND_SHADER_RESOURCE;
```

In addition a shader resource view needs to be created for the depth buffer using a [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure and

[ID3D11Device::CreateShaderResourceView](#). The shader resource view will use a typed format such as `DXGI_FORMAT_R32_FLOAT` that is the equivalent to the typeless format specified when the depth-stencil resource was created.

In the first render pass the depth buffer is bound as described in the [Bind Depth-Stencil Data to the OM Stage](#) section. Note that the format passed to [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#).Format will use a typed format such as `DXGI_FORMAT_D32_FLOAT`. After the first render pass the depth buffer will contain the depth values for the scene.

In the second render pass the [ID3D11DeviceContext::OMSetRenderTarget](#)s function is used to set the depth-stencil view to `NULL` or a different depth-stencil resource and the shader resource view is passed to the shader using [ID3D11EffectShaderResourceVariable::SetResource](#). This allows the shader to look up the depth values calculated in the first rendering pass. Note that a transformation will

need to be applied to retrieve depth values if the point of view of the first render pass is different from the second render pass. For example, if a shadow mapping technique is being used the first render pass will be from the perspective of a light source while the second render pass will be from the viewer's perspective.

## Related topics

[Output-Merger Stage](#)

[Pipeline Stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Configuring Blending Functionality

Article • 08/19/2020

Blending operations are performed on every pixel shader output (RGBA value) before the output value is written to a render target. If multisampling is enabled, blending is done on each multisample; otherwise, blending is performed on each pixel.

- [Create the Blend State](#)
- [Bind the Blend State](#)
- [Advanced Blending Topics](#)
  - [Alpha-To-Coverage](#)
  - [Blending Pixel Shader Outputs](#)
- [Related topics](#)

## Create the Blend State

The blend state is a collection of states used to control blending. These states (defined in [D3D11\\_BLEND\\_DESC1](#)) are used to create the blend state object by calling [ID3D11Device1::CreateBlendState1](#).

For instance, here is a very simple example of blend-state creation that disables alpha blending and uses no per-component pixel masking.

```
ID3D11BlendState1* g_pBlendStateNoBlend = NULL;

D3D11_BLEND_DESC1 BlendState;
ZeroMemory(&BlendState, sizeof(D3D11_BLEND_DESC1));
BlendState.RenderTarget[0].BlendEnable = FALSE;
BlendState.RenderTarget[0].RenderTargetWriteMask =
D3D11_COLOR_WRITE_ENABLE_ALL;
pd3dDevice->CreateBlendState1(&BlendState, &g_pBlendStateNoBlend);
```

This example is similar to the [HLSLWithoutFX10 Sample](#).

## Bind the Blend State

After you create the blend-state object, bind the blend-state object to the output-merger stage by calling [ID3D11DeviceContext::OMSetBlendState](#).

```
float blendFactor[4] = { 0.0f, 0.0f, 0.0f, 0.0f };
UINT sampleMask    = 0xffffffff;

pd3dDevice->OMSetBlendState(g_pBlendStateNoBlend, blendFactor, sampleMask);
```

This API takes three parameters: the blend-state object, a four-component blend factor, and a sample mask. You can pass in **NULL** for the blend-state object to specify the default blend state or pass in a blend-state object. If you created the blend-state object with **D3D11\_BLEND\_BLEND\_FACTOR** or **D3D11\_BLEND\_INV\_BLEND\_FACTOR**, you can pass a blend factor to modulate values for the pixel shader, render target, or both. If you didn't create the blend-state object with **D3D11\_BLEND\_BLEND\_FACTOR** or **D3D11\_BLEND\_INV\_BLEND\_FACTOR**, you can still pass a non-NULL blend factor, but the blending stage does not use the blend factor; the runtime stores the blend factor, and you can later call **ID3D11DeviceContext::OMGetBlendState** to retrieve the blend factor. If you pass **NULL**, the runtime uses or stores a blend factor equal to { 1, 1, 1, 1 }. The sample mask is a user-defined mask that determines how to sample the existing render target before updating it. The default sampling mask is 0xffffffff which designates point sampling.

In most depth buffering schemes, the pixel closest to the camera is the one that gets drawn. When [setting up the depth stencil state](#), the **DepthFunc** member of **D3D11\_DEPTH\_STENCIL\_DESC** can be any **D3D11\_COMPARISON\_FUNC**. Normally, you would want **DepthFunc** to be **D3D11\_COMPARISON\_LESS**, so that the pixels closest to the camera will overwrite the pixels behind them. However, depending on the needs of your application, any of the other comparison functions may be used to do the depth test.

## Advanced Blending Topics

- [Alpha-To-Coverage](#)
- [Blending Pixel Shader Outputs](#)

### Alpha-To-Coverage

Alpha-to-coverage is a multisampling technique that is most useful for situations such as dense foliage where there are several overlapping polygons that use alpha transparency to define edges within the surface.

You can use the **AlphaToCoverageEnable** member of **D3D11\_BLEND\_DESC1** or **D3D11\_BLEND\_DESC** to toggle whether the runtime converts the .a component (alpha) of output register **SV\_Target0** from the pixel shader to an n-step coverage mask (given

an n-sample **RenderTarget**). The runtime performs an **AND** operation of this mask with the typical sample coverage for the pixel in the primitive (in addition to the sample mask) to determine which samples to update in all the active **RenderTarget**s.

If the pixel shader outputs **SV\_Coverage**, the runtime disables alpha-to-coverage.

### ⓘ Note

In multisampling, the runtime shares only one coverage for all **RenderTarget**s. The fact that the runtime reads and converts .a from output **SV\_Target0** to coverage when **AlphaToCoverageEnable** is TRUE does not change the .a value that goes to the blender at **RenderTarget** 0 (if a **RenderTarget** happens to be set there). In general, if you enable alpha-to-coverage, you don't affect how all color outputs from pixel shaders interact with **RenderTarget**s through the **output-merger stage** except that the runtime performs an **AND** operation of the coverage mask with the alpha-to-coverage mask. Alpha-to-coverage works independently to whether the runtime can blend **RenderTarget** or whether you use blending on **RenderTarget**.

Graphics hardware doesn't precisely specify exactly how it converts pixel shader **SV\_Target0.a** (alpha) to a coverage mask, except that alpha of 0 (or less) must map to no coverage and alpha of 1 (or greater) must map to full coverage (before the runtime performs an **AND** operation with actual primitive coverage). As alpha goes from 0 to 1, the resulting coverage should generally increase monotonically. However, hardware might perform area dithering to provide some better quantization of alpha values at the cost of spatial resolution and noise. An alpha value of NaN (Not a Number) results in a no coverage (zero) mask.

Alpha-to-coverage is also traditionally used for screen-door transparency or defining detailed silhouettes for otherwise opaque sprites.

## Blending Pixel Shader Outputs

This feature enables the output merger to use both the pixel shader outputs simultaneously as input sources to a blending operation with the single render target at slot 0.

This example takes two results and combines them in a single pass, blending one into the destination with a multiply and the other with an add:

```
SrcBlend = D3D11_BLEND_ONE;  
DestBlend = D3D11_BLEND_SRC1_COLOR;
```

This example configures the first pixel shader output as the source color and the second output as a per-color component blend factor.

```
SrcBlend = D3D11_BLEND_SRC1_COLOR;  
DestBlend = D3D11_BLEND_INV_SRC1_COLOR;
```

This example illustrates how the blend factors must match the shader swizzles:

```
SrcFactor = D3D11_BLEND_SRC1_ALPHA;  
DestFactor = D3D11_BLEND_SRC_COLOR;  
OutputWriteMask[0] = .ra; // pseudocode for setting the mask at  
// RenderTarget slot 0 to .ra
```

Together, the blend factors and the shader code imply that the pixel shader is required to output at least o0.r and o1.a. Extra output components can be output by the shader but would be ignored, fewer components would produce undefined results.

## Related topics

[Output-Merger Stage](#)

[Pipeline Stages \(Direct3D 10\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Depth Bias

Article • 11/04/2020

Polygons that are coplanar in 3D space can be made to appear as if they are not coplanar by adding a z-bias (or depth bias) to each one.

This is a technique commonly used to ensure that shadows in a scene are displayed properly. For instance, a shadow on a wall will likely have the same depth value as the wall. If an application renders a wall first and then a shadow, the shadow might not be visible, or depth artifacts might be visible.

An application can help ensure that coplanar polygons are rendered properly by adding the bias (from the **DepthBias** member of [D3D11\\_RASTERIZER\\_DESC1](#)) to the z-values that the system uses when rendering the sets of coplanar polygons. Polygons with a larger z value will be drawn in front of polygons with a smaller z value.

There are two options for calculating depth bias.

1. If the depth buffer currently bound to the output-merger stage has a **UNORM** format or no depth buffer is bound, the bias value is calculated like this:

```
Bias = (float)DepthBias * r + SlopeScaledDepthBias * MaxDepthSlope;
```

where  $r$  is the minimum representable value  $> 0$  in the depth-buffer format converted to **float32**. The **DepthBias** and **SlopeScaledDepthBias** values are [D3D11\\_RASTERIZER\\_DESC1](#) structure members. The **MaxDepthSlope** value is the maximum of the horizontal and vertical slopes of the depth value at the pixel.

2. If a floating-point depth buffer is bound to the output-merger stage the bias value is calculated like this:

```
Bias = (float)DepthBias * 2**(exponent(max z in primitive) - r) +  
SlopeScaledDepthBias * MaxDepthSlope;
```

where  $r$  is the number of mantissa bits in the floating point representation (excluding the hidden bit); for example, 23 for **float32**.

The bias value is then clamped like this:

```
if(DepthBiasClamp > 0)
    Bias = min(DepthBiasClamp, Bias)
else if(DepthBiasClamp < 0)
    Bias = max(DepthBiasClamp, Bias)
```

The bias value is then used to calculate the pixel depth.

```
if ( (DepthBias != 0) || (SlopeScaledDepthBias != 0) )
    z = z + Bias
```

Depth-bias operations occur on vertices after clipping, therefore, depth-bias has no effect on geometric clipping. The bias value is constant for a given primitive and is added to the z value for each vertex before interpolator setup. When you use [feature levels](#) 10.0 and higher, all bias calculations are performed using 32-bit floating-point arithmetic. Bias is not applied to any point or line primitives, except for lines drawn in wireframe mode.

One of the artifacts with shadow buffer based shadows is shadow acne, or a surface shadowing itself due to minor differences between the depth computation in a shader, and the depth of the same surface in the shadow buffer. One way to alleviate this is to use **DepthBias** and **SlopeScaledDepthBias** when rendering a shadow buffer. The idea is to push surfaces out enough while rendering a shadow buffer so that the comparison result (between the shadow buffer z and the shader z) is consistent across the surface, and avoid local self-shadowing.

However, using **DepthBias** and **SlopeScaledDepthBias** can introduce new rendering problems when a polygon viewed at an extremely sharp angle causes the bias equation to generate a very large z value. This in effect pushes the polygon extremely far away from the original surface in the shadow map. One way to help alleviate this particular problem is to use **DepthBiasClamp**, which provides an upper bound (positive or negative) on the magnitude of the z bias calculated.

#### ⓘ Note

For feature levels 9.1, 9.2, 9.3, **DepthBiasClamp** is not supported.

# Related topics

[Output-Merger Stage](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Compute Shader Overview

Article • 04/09/2021

A compute shader is a programmable shader stage that expands Microsoft Direct3D 11 beyond graphics programming. The compute shader technology is also known as the DirectCompute technology.

Like other programmable shaders (vertex and geometry shaders for example), a compute shader is designed and implemented with [HLSL](#) but that is just about where the similarity ends. A compute shader provides high-speed general purpose computing and takes advantage of the large numbers of parallel processors on the graphics processing unit (GPU). The compute shader provides memory sharing and thread synchronization features to allow more effective parallel programming methods. You call the [ID3D11DeviceContext::Dispatch](#) or [ID3D11DeviceContext::DispatchIndirect](#) method to execute commands in a compute shader. A compute shader can run on many threads in parallel.

## Using Compute Shader on Direct3D 10.x Hardware

A compute shader on Microsoft Direct3D 10 is also known as DirectCompute 4.x.

If you use the Direct3D 11 API and updated drivers, [feature level](#) 10 and 10.1 Direct3D hardware can optionally support a limited form of DirectCompute that uses the cs\_4\_0 and cs\_4\_1 [profiles](#). When you use DirectCompute on this hardware, keep the following limitations in mind:

- The maximum number of threads is limited to D3D11\_CS\_4\_X\_THREAD\_GROUP\_MAX\_THREADS\_PER\_GROUP (768) per group.
- The X and Y dimension of **numthreads** is limited to D3D11\_CS\_4\_X\_THREAD\_GROUP\_MAX\_X (768) and D3D11\_CS\_4\_X\_THREAD\_GROUP\_MAX\_Y (768).
- The Z dimension of **numthreads** is limited to 1.
- The Z dimension of dispatch is limited to D3D11\_CS\_4\_X\_DISPATCH\_MAX\_THREAD\_GROUPS\_IN\_Z\_DIMENSION (1).
- Only one unordered-access view can be bound to the shader (D3D11\_CS\_4\_X\_UAV\_REGISTER\_COUNT is 1).
- Only [RWStructuredBuffers](#) and [RWByteAddressBuffers](#)s are available as unordered-access views.

- A thread can only access its own region in groupshared memory for writing, though it can read from any location.
- [SV\\_GroupIndex](#) or [SV\\_GroupThreadID](#) must be used when accessing **groupshared** memory for writing.
- **Groupshared** memory is limited to 16KB per group.
- A single thread is limited to a 256 byte region of **groupshared** memory for writing.
- No atomic instructions are available.
- No double-precision values are available.

## Using Compute Shader on Direct3D 11.x Hardware

A compute shader on Direct3D 11 is also known as DirectCompute 5.0.

When you use DirectCompute with [cs\\_5\\_0 profiles](#), keep the following items in mind:

- The maximum number of threads is limited to `D3D11_CS_THREAD_GROUP_MAX_THREADS_PER_GROUP` (1024) per group.
- The X and Y dimension of **numthreads** is limited to `D3D11_CS_THREAD_GROUP_MAX_X` (1024) and `D3D11_CS_THREAD_GROUP_MAX_Y` (1024).
- The Z dimension of **numthreads** is limited to `D3D11_CS_THREAD_GROUP_MAX_Z` (64).
- The maximum dimension of dispatch is limited to `D3D11_CS_DISPATCH_MAX_THREAD_GROUPS_PER_DIMENSION` (65535).
- The maximum number of unordered-access views that can be bound to a shader is `D3D11_PS_CS_UAV_REGISTER_COUNT` (8).
- Supports [RWStructuredBuffers](#), [RWByteAddressBuffers](#), and typed unordered-access views ([RWTexture1D](#), [RWTexture2D](#), [RWTexture3D](#), and so on).
- Atomic instructions are available.
- Double-precision support might be available. For information about how to determine whether double-precision is available, see [D3D11\\_FEATURE\\_DOUBLES](#).

## In this section

Topic	Description
<a href="#">New Resource Types</a>	Several new resource types have been added in Direct3D 11.

Topic	Description
Accessing Resources	There are several ways to access <a href="#">resources</a> .
Atomic Functions	To access a new resource type or shared memory, use an interlocked intrinsic function. Interlocked functions are guaranteed to operate atomically. That is, they are guaranteed to occur in the order programmed. This section lists the atomic functions.

## Related topics

[Graphics Pipeline](#)

[How To: Create a Compute Shader](#)

---

## Feedback

Was this page helpful?



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

# New Resource Types

Article • 08/19/2020

Several new resource types have been added in Direct3D 11.

- [Read/Write Buffers and Textures](#)
- [Structured Buffer](#)
- [Byte Address Buffer](#)
- [Unordered Access Buffer or Texture](#)
  - [Append and Consume Buffer](#)
- [Related topics](#)

## Read/Write Buffers and Textures

Shader model 4 resources are read only. Shader model 5 implements a new corresponding set of read/write resources:

- [RWBuffer](#)
- [RWTexture1D, RWTexture1DArray](#)
- [RWTexture2D, RWTexture2DArray](#)
- [RWTexture3D](#)

These resources require a resource variable to access memory (through indexing) as there are no methods for accessing memory directly. A resource variable can be used on the left and right sides of an equation; if used on the right side, the template type must be single component (float, int, or uint).

## Structured Buffer

A structured buffer is a buffer that contains elements of equal sizes. Use a structure with one or more member types to define an element. Here is a structure with three members.

```
struct MyStruct
{
    float4 Color;
    float4 Normal;
    bool isAwesome;
};
```

Use this structure to declare a structured buffer like this:

```
StructuredBuffer<MyStruct> mySB;
```

In addition to indexing, a structured buffer supports accessing a single member like this:

```
float4 myColor = mySb[27].Color;
```

Use the following object types to access a structured buffer:

- [StructuredBuffer](#) is a read only structured buffer.
- [RWStructuredBuffer](#) is a read/write structured buffer.

[Atomic functions](#) which implement interlocked operations are allowed on int and uint elements in an [RWStructuredBuffer](#).

## Byte Address Buffer

A byte address buffer is a buffer whose contents are addressable by a byte offset. Normally, the contents of a [buffer](#) are indexed per element using a stride for each element (S) and the element number (N) as given by S\*N. A byte address buffer, which can also be called a raw buffer, uses a byte value offset from the beginning of the buffer to access data. The byte value must be a multiple of four so that it is DWORD aligned. If any other value is provided, behavior is undefined.

Shader model 5 introduces objects for accessing a [read-only byte address buffer](#) as well as a [read-write byte address buffer](#). The contents of a byte address buffer is designed to be a 32-bit unsigned integer; if the value in the buffer is not really an unsigned integer, use a function such as [asfloat](#) to read it.

## Unordered Access Buffer or Texture

An unordered access resource (which includes buffers, textures, and texture arrays - without multisampling), allows temporally unordered read/write access from multiple threads. This means that this resource type can be read/written simultaneously by multiple threads without generating memory conflicts through the use of [Atomic Functions](#).

Create an unordered access buffer or texture by calling a function such as [ID3D11Device::CreateBuffer](#) or [ID3D11Device::CreateTexture2D](#) and passing in the `D3D11_BIND_UNORDERED_ACCESS` flag from the [D3D11\\_BIND\\_FLAG](#) enumeration.

Unordered access resources can only be bound to pixel shaders and compute shaders. During execution, pixel shaders or compute shaders running in parallel have the same unordered access resources bound.

## Append and Consume Buffer

An append and consume buffer is a special type of an unordered resource that supports adding and removing values from the end of a buffer similar to the way a stack works.

An append and consume buffer must be a structured buffer:

- [AppendStructuredBuffer](#)
- [ConsumeStructuredBuffer](#)

Use these resources through their methods, these resources do not use resource variables.

## Related topics

[Compute Shader Overview](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Accessing Resources

Article • 08/19/2020

There are several ways to access [resources](#). Regardless, Direct3D guarantees to return zero for any resource that is accessed out of bounds.

- [Access By Byte Offset](#)
- [Access By Index](#)
- [Access By Mips Method](#)
- [Related topics](#)

## Access By Byte Offset

Two new buffer types can be accessed using a byte offset:

- [ByteAddressBuffer](#) is a read-only buffer.
- [RWByteAddressBuffer](#) is a read or write buffer.

## Access By Index

Resource types can use an index to reference a specific location in the resource.

Consider this example:

```
uint2 pos;
Texture2D<float4> myTexture;
float4 myVar = myTexture[pos];
```

This example assigns the 4 float values that are stored at the texel located at the *pos* position in the *myTexture* 2D texture resource to the *myVar* variable.

### ⓘ Note

The default for accessing a texture in this way is mipmap level zero (the most detailed level).

### ⓘ Note

The "float4 myVar = myTexture[pos];" line is equivalent to "float4 myVar = myTexture.Load(uint3(pos,0));". Access by index is a new HLSL syntax enhancement.

#### ⓘ Note

The compiler in the June 2010 version of the DirectX SDK and later lets you index all resource types except for **byte address buffers**.

#### ⓘ Note

The June 2010 compiler and later lets you declare local resource variables. You can assign globally defined resources (like *myTexture*) to these variables and use them the same way as their global counterparts.

## Access By Mips Method

Texture objects have a **mips** method (for example, [Texture2D.mips](#)), which you can use to specify the mipmap level. This example reads the color stored at (7,16) on mipmap level 2 in a 2D texture:

```
uint x = 7;
uint y = 16;
float4 myColor = myTexture.mips[2][uint2(x,y)];
```

This is an enhancement from the June 2010 compiler and later. The "myTexture.mips[2][uint2(x,y)]" expression is equivalent to "myTexture.Load(uint3(x,y,2))".

## Related topics

[Compute Shader Overview](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Atomic Functions

Article • 08/19/2020

To access a new resource type or shared memory, use an interlocked intrinsic function. Interlocked functions are guaranteed to operate atomically. That is, they are guaranteed to occur in the order programmed. This section lists the atomic functions.

- [InterlockedAdd](#)
- [InterlockedMin](#)
- [InterlockedMax](#)
- [InterlockedOr](#)
- [InterlockedAnd](#)
- [InterlockedXor](#)
- [InterlockedCompareStore](#)
- [InterlockedCompareExchange](#)
- [InterlockedExchange](#)

## Related topics

[Compute Shader Overview](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Rendering (Direct3D 11 Graphics)

Article • 12/10/2020

This section contains information about several Direct3D 11 rendering technologies.

## In this section

Topic	Description
<a href="#">MultiThreading</a>	Direct3D 11 implements support for object creation and rendering using multiple threads.
<a href="#">Multiple-Pass Rendering</a>	Multiple-pass rendering is a process in which an application traverses its scene graph multiple times in order to produce an output to render to the display. Multiple-pass rendering improves performance because it breaks up complex scenes into tasks that can run concurrently.

## Related topics

[Programming Guide for Direct3D 11](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# MultiThreading

Article • 08/23/2019

Direct3D 11 implements support for object creation and rendering using multiple threads.

## In this section

Topic	Description
<a href="#">Introduction to Multithreading in Direct3D 11</a>	Multithreading is designed to improve performance by performing work using one or more threads at the same time.
<a href="#">Object Creation with Multithreading</a>	Use the <a href="#">ID3D11Device</a> interface to create resources and objects, use the <a href="#">ID3D11DeviceContext</a> for <a href="#">rendering</a> .
<a href="#">Immediate and Deferred Rendering</a>	Direct3D 11 supports two types of rendering: immediate and deferred. Both are implemented by using the <a href="#">ID3D11DeviceContext</a> interface.
<a href="#">Command List</a>	A command list is a sequence of GPU commands that can be recorded and played back. A command list may improve performance by reducing the amount of overhead generated by the runtime.
<a href="#">Threading Differences between Direct3D Versions</a>	Many multi-threaded programming models make use of synchronization primitives (such as mutexes) to create critical sections and prevent code from being accessed by more than one thread at a time.

## Related topics

[How To: Check for Driver Support](#)

[Rendering](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Introduction to Multithreading in Direct3D 11

Article • 08/19/2020

Multithreading is designed to improve performance by performing work using one or more threads at the same time.

In the past, this has often been done by generating a single main thread for rendering and one or more threads for doing preparation work such as object creation, loading, processing, and so on. However, with the built in synchronization in Direct3D 11, the goal behind multithreading is to utilize every CPU and GPU cycle without making a processor wait for another processor (particularly not making the GPU wait because it directly impacts frame rate). By doing so, you can generate the most amount of work while maintaining the best frame rate. The concept of a single frame for rendering is no longer as necessary since the API implements synchronization.

Multithreading requires some form of synchronization. For example, if multiple threads that run in an application must access a single device context ([ID3D11DeviceContext](#)), that application must use some synchronization mechanism, such as critical sections, to synchronize access to that device context. This is because processing of the render commands (generally done on the GPU) and generating the render commands (generally done on the CPU through object creation, data loading, state changing, data processing) often use the same resources (textures, shaders, pipeline state, and so on). Organizing the work across multiple threads requires synchronization to prevent one thread from modifying or reading data that is being modified by another thread.

While the use of a device context ([ID3D11DeviceContext](#)) is not thread-safe, the use of a Direct3D 11 device ([ID3D11Device](#)) is thread-safe. Because each [ID3D11DeviceContext](#) is single threaded, only one thread can call a [ID3D11DeviceContext](#) at a time. If multiple threads must access a single [ID3D11DeviceContext](#), they must use some synchronization mechanism, such as critical sections, to synchronize access to that [ID3D11DeviceContext](#). However, multiple threads are not required to use critical sections or synchronization primitives to access a single [ID3D11Device](#). Therefore, if an application uses [ID3D11Device](#) to create resource objects, that application is not required to use synchronization to create multiple resource objects at the same time.

Multithreading support divides the API into two distinct functional areas:

- [Object Creation with Multithreading](#)
- [Immediate and Deferred Rendering](#)

Multithreading performance depends on the driver support. [How To: Check for Driver Support](#) provides more information about querying the driver and what the results mean.

Direct3D 11 has been designed from the ground up to support multithreading. Direct3D 10 implements limited support for multithreading using the [thread-safe layer](#). This page lists the behavior differences between the two versions of DirectX: [Threading Differences between Direct3D Versions](#).

## Multithreading and DXGI

Only one thread at a time should use the immediate context. However, your application should also use that same thread for Microsoft DirectX Graphics Infrastructure (DXGI) operations, especially when the application makes calls to the [IDXGISwapChain::Present](#) method.

 **Note**

It is invalid to use an immediate context concurrently with most of the DXGI interface functions. For the March 2009 and later DirectX SDKs, the only DXGI functions that are safe are [AddRef](#), [Release](#), and [QueryInterface](#).

For more info about using DXGI with multiple threads, see [Multithread Considerations](#).

## Related topics

[Multithreading](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Object Creation with Multithreading

Article • 08/23/2019

Use the [ID3D11Device](#) interface to create resources and objects, use the [ID3D11DeviceContext](#) for rendering.

Here are some examples of how to create resources:

- [How to: Create a Vertex Buffer](#)
- [How to: Create an Index Buffer](#)
- [How to: Create a Constant Buffer](#)
- [How to: Initialize a Texture From a File](#)

## Multithreading Considerations

The amount of concurrency of resource creation and rendering that your application can achieve depends on the level of multithreading support that the driver implements. If the driver supports concurrent creates, then the application should have much less concurrency concerns. However, if the driver does not support concurrent object creation, the amount of concurrency is extremely limited. To understand the amount of support available in a driver, query the driver for the value of the [DriverConcurrentCreates](#) member of the [D3D11\\_FEATURE\\_DATA\\_THREADING](#) structure. For more information about checking for driver support of multithreading, see [How To: Check for Driver Support](#).

Concurrent operations do not necessarily lead to better performance. For example, creating and loading a texture is typically limited by memory bandwidth. Attempting to create and load multiple textures might be no faster than doing one texture at a time, even if this leaves multiple CPU cores idle. However, creating multiple shaders using multiple threads can increase performance as this operation is less dependent on system resources such as memory bandwidth.

When the driver and graphics hardware support multithreading, you can typically initialize newly created resources more efficiently by passing a pointer to initial data in the *pInitialData* parameter (for example, in a call to the [ID3D11Device::CreateTexture2D](#) method).

## Related topics

[MultiThreading](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Immediate and Deferred Rendering

Article • 08/23/2019

Direct3D 11 supports two types of rendering: immediate and deferred. Both are implemented by using the [ID3D11DeviceContext](#) interface.

## Immediate Rendering

Immediate rendering refers to calling rendering APIs or commands from a device, which queues the commands in a buffer for execution on the GPU. Use an immediate context to render, set pipeline state, and play back a command list.

Create an immediate context using [D3D11CreateDevice](#) or [D3D11CreateDeviceAndSwapChain](#).

## Deferred Rendering

Deferred rendering records graphics commands in a command buffer so that they can be played back at some other time. Use a deferred context to record commands (rendering as well as state setting) to a command list. Deferred rendering is a new concept in Direct3D 11; deferred rendering is designed to support rendering on one thread while recording commands for rendering on additional threads. This parallel, multithread strategy allows you to break up complex scenes into concurrent tasks. For more information about rendering complex scenes, see [Multiple-Pass Rendering](#).

Create a deferred context using [ID3D11Device::CreateDeferredContext](#).

Direct3D generates rendering overhead when it queues up commands in the command buffer. In contrast, a [command list](#) executes much more efficiently during playback. To use a command list, record rendering commands with a deferred context and play them back using an immediate context.

You can generate only a single command list in a single-threaded fashion. However, you can create and use multiple deferred contexts simultaneously, each in a separate thread. Then, you can use those multiple deferred contexts to simultaneously create multiple command lists.

You cannot play back two or more command lists simultaneously on the immediate context.

To determine if a device context is an immediate or a deferred context, call [ID3D11DeviceContext::GetType](#).

The [ID3D11DeviceContext::Map](#) method is only supported in a deferred context for dynamic resources ([D3D11\\_USAGE\\_DYNAMIC](#)) where the first call within the command list is [D3D11\\_MAP\\_WRITE\\_DISCARD](#). [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#) is supported on subsequent calls if available for the given type of resource.

Queries in a deferred context are limited to data generation and predicated drawing. You cannot call [ID3D11DeviceContext::GetData](#) on a deferred context to get data about a query; you can only call [GetData](#) on the immediate context to get data about a query. You can set a rendering predicate (a type of query) by calling [D3D11DeviceContext::SetPredication](#) to use query results on the GPU. You can generate query data through calls to [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). However, the query data will not be available until you call [ID3D11DeviceContext::ExecuteCommandList](#) on the immediate context to submit the deferred context command list. The query data is then processed by the GPU.

## Related topics

[Multithreading](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Command List

Article • 08/23/2019

A command list is a sequence of GPU commands that can be recorded and played back. A command list may improve performance by reducing the amount of overhead generated by the runtime.

Use a command list in the following scenarios:

- Within a single frame, render part of the scene on one thread while recording another part of the scene on a second thread. At the end of the frame, play back the recorded command list on the first thread. Use this approach to scale complex rendering tasks across multiple threads or cores.
- Pre-record a command list before you need to render it (for example, while a level is loading) and efficiently play it back later in your scene. This optimization works well when you need to render something often.

A command list is immutable and is designed to be recorded and played back during a single execution of an application. A command list is not designed to be pre-recorded ahead of game execution and loaded from your media as there is no way to persist the list.

A command list must be recorded by a deferred context, but it can only be played back on an immediate context. Deferred contexts can generate command lists concurrently.

- To record a command list, see [How to: Record a Command List](#).
- To play back a command list, see [How to: Play Back a Command List](#).
- When using a command list, performance depends on the amount of support implemented in a driver. To check for driver support, see [How To: Check for Driver Support](#).

## Related topics

[Immediate and Deferred Rendering](#)

[MultiThreading](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# Threading Differences between Direct3D Versions

Article • 08/19/2020

Many multi-threaded programming models make use of synchronization primitives (such as mutexes) to create critical sections and prevent code from being accessed by more than one thread at a time.

However, the resource creation methods of the [ID3D11Device](#) interface were designed to be re-entrant, without requiring synchronization primitives and critical sections. As a result, the resource creation methods are efficient and easy to use.

 Expand table

Differences between Direct3D 11, 10 and 9:

Direct3D 11 defaults to mostly thread-safe and continues to allow applications to opt-out using `D3D11_CREATE_DEVICE_SINGLETHREADED`. If applications opt-out of being thread-safe, they must adhere to threading rules. The runtime synchronizes threads on behalf of the application allowing concurrent threads to run. In fact, the synchronization in Direct3D 11 is more efficient than using the thread-safe layer in Direct3D 10.

Direct3D 10 can support the execution of only one thread at a time. Direct3D 10 is fully thread safe and allows an application to opt-out of that behavior by using `D3D10_CREATE_DEVICE_SINGLE_THREADED`.

Direct3D 9 does not default to thread safe. However, when you call [CreateDevice](#) or [CreateDeviceEx](#) to create a device, you can specify the `D3DCREATE_MULTITHREADED` flag to make the Direct3D 9 API thread safe. This causes significant synchronization overhead. Therefore, making the Direct3D 9 API thread safe is not recommended because performance can be degraded.

## Related topics

[MultiThreading](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Multiple-Pass Rendering

Article • 08/23/2019

Multiple-pass rendering is a process in which an application traverses its scene graph multiple times in order to produce an output to render to the display. Multiple-pass rendering improves performance because it breaks up complex scenes into tasks that can run concurrently.

To perform multiple-pass rendering, you create a deferred context and command list for each additional pass. While the application traverses the scene graph, it records commands (for example, rendering commands such as [Draw](#)) into a deferred context. After the application finishes the traversal, it calls the [FinishCommandList](#) method on the deferred context. Finally, the application calls the [ExecuteCommandList](#) method on the immediate context to execute the commands in each command list.

The following pseudocode shows how to perform multiple-pass rendering:

```
syntax

{
    ImmCtx->SetRenderTarget( pRTViewOfResourceX );
    DefCtx1->SetTexture( pSRView1OfResourceX );
    DefCtx2->SetTexture( pSRView2OfResourceX );

    for () // Traverse the scene graph.
    {
        ImmCtx->Draw(); // Pass 0: immediate context renders primitives into
        resource X.

        // The following texturing by the deferred contexts occurs after the
        // immediate context makes calls to ExecuteCommandList.
        // Resource X is then completely updated by the immediate context.
        DefCtx1->Draw(); // Pass 1: deferred context 1 performs texturing from
        resource X.
        DefCtx2->Draw(); // Pass 2: deferred context 2 performs texturing from
        resource X.
    }

    // Create command lists and record commands into them.
    DefCtx1->FinishCommandList( &pCL1 );
    DefCtx2->FinishCommandList( &pCL2 );

    ImmCtx->ExecuteCommandList( pCL1 ); // Execute pass 1.
    ImmCtx->ExecuteCommandList( pCL2 ); // Execute pass 2.
}
```

### Note

The immediate context modifies a resource, which is bound to the immediate context as a render target view (RTV); in contrast, each deferred context simply uses the resource, which is bound to the deferred context as a shader resource view (SRV). For more information about immediate and deferred contexts, see [Immediate and Deferred Rendering](#).

## Related topics

[Rendering](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Effects (Direct3D 11)

Article • 06/11/2021

A DirectX effect is a collection of pipeline state, set by expressions written in [HLSL](#) and some syntax that is specific to the effect framework.

After compiling an effect, use the effect framework APIs to render. Effect functionality can range from something as simple as a vertex shader that transforms geometry and a pixel shader that outputs a solid color, to a rendering technique that requires multiple passes, uses every stage of the graphics pipeline, and manipulates shader state as well as the pipeline state not associated with the programmable shaders.

The first step is to organize the state you want to control in an effect. This includes shader state (vertex, hull, domain, geometry, pixel and compute shaders), texture and sampler state used by the shaders, and other non-programmable pipeline state. You can create an effect in memory as a text string, but typically, the size gets large enough that it is handy to store effect state in an effect file (a text file that ends in a .fx extension). To use an effect, you must compile it (to check HLSL syntax as well as effect framework syntax), initialize effect state through API calls, and modify your render loop to call the rendering APIs.

An effect encapsulates all of the render state required by a particular effect into a single rendering function called a technique. A pass is a sub-set of a technique, that contains render state. To implement a multiple pass rendering effect, implement one or more passes within a technique. For example, say you wanted to render some geometry with one set of depth/stencil buffers, and then draw some sprites on top of that. You could implement the geometry rendering in the first pass, and the sprite drawing in the second pass. To render the effect, you simply render both passes in your render loop. You can implement any number of techniques in an effect. Of course, the greater the number of techniques, the greater the compile time for the effect. One way to exploit this functionality is to create effects with techniques that are designed to run on different hardware. This allows an application to gracefully downgrade performance based on the hardware capabilities detected.

A set of techniques can be grouped in a group (which uses the syntax "fxgroup"). Techniques can be grouped in any way. For example, multiple groups could be created, one per material; each material could have a technique for each hardware level; each technique would have a set of passes which define the material on the particular hardware.

# In this section

Topic	Description
<a href="#">Organizing State in an Effect</a>	With Direct3D 11, effect state for certain pipeline stages is organized by structures.
<a href="#">Effect System Interfaces</a>	The effect system defines several interfaces for managing effect state.
<a href="#">Specializing Interfaces</a>	<a href="#">ID3DX11EffectVariable</a> has a number of methods for casting the interface into the particular type of interface you need.
<a href="#">Interfaces and Classes in Effects</a>	There are many ways to use classes and interfaces in Effects 11.
<a href="#">Rendering an Effect</a>	An effect can be used to store information, or to render using a group of state.
<a href="#">Cloning an Effect</a>	Cloning an effect creates a second, almost identical copy of the effect.
<a href="#">Stream Out Syntax</a>	A geometry shader with stream out is declared with a particular syntax.
<a href="#">Differences Between Effects 10 and Effects 11</a>	This topic shows the differences between Effects 10 and Effects 11.

## Related topics

[Programming Guide for Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Organizing State in an Effect (Direct3D 11)

Article • 08/19/2020

With Direct3D 11, effect state for certain pipeline stages is organized by structures. Here are the structures:

Pipeline State	Structure
Rasterization	<a href="#">D3D11_RASTERIZER_DESC</a>
Output Merger	<a href="#">D3D11_BLEND_DESC</a> and <a href="#">D3D11_DEPTH_STENCIL_DESC</a>
Shaders	See below

For the shader stages, where the number of state changes need to be more controlled by an application, the state has been divided up into constant buffer state, sampler state, shader resource state, and unordered access view state (for pixel and compute shaders). This allows an application that is carefully designed to update only the state that is changing, which improves performance by reducing the amount of data that needs to be passed to the GPU.

So how do you organize the pipeline state in an effect?

The answer is, the order doesn't matter. Global variables do not have to be located at the top. However, all the samples in the SDK follow the same order, as it is good practice to organize the data the same way. So this is a brief description of the data ordering in the DirectX SDK samples.

- [Global Variables](#)
- [Shaders](#)
- [Groups, Techniques, and Passes](#)

## Global Variables

Just like standard C practice, global variables are declared first, at the top of the file. Most often, these are variables that will be initialized by an application, and then used in an effect. Sometimes they are initialized and never changed, other times they are updated every frame. Just like C function scope rules, effect variables declared outside

of the scope of effect functions are visible throughout the effect; any variable declared inside of an effect function is only visible within that function.

Here is an example of the variables declared in BasicHLSL10.fx.

```
// Global variables
float4 g_MaterialAmbientColor;           // Material's ambient color

Texture2D g_MeshTexture;                  // Color texture for mesh

float    g_fTime;                        // App's time in seconds
float4x4 g_mWorld;                      // World matrix for object
float4x4 g_mWorldViewProjection;         // World * View * Projection matrix

// Texture samplers
SamplerState MeshTextureSampler
{
    Filter = MIN_MAG_MIP_LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};
```

The syntax for effect variables is more fully detailed in [Effect Variable Syntax \(Direct3D 11\)](#). The syntax for effect texture samplers is more fully detailed in [Sampler Type \(DirectX HLSL\)](#).

## Shaders

Shaders are small executable programs. You can think of shaders as encapsulating shader state, since the HLSL code implements the shader functionality. The graphics pipeline up to five different kinds of shaders.

- Vertex shaders - Operate on vertex data. One vertex in yields one vertex out.
- Hull shaders - Operate on patch data. Control Point Phase: one invocation yields one control point; For each Fork and Join Phases: one patch yields some amount of patch constant data.
- Domain shaders - Operate on primitive data. One primitive may yield 0, 1, or many primitives out.
- Geometry shaders - Operate on primitive data. One primitive in may yield 0, 1, or many primitives out.
- Pixel shaders - Operate on pixel data. One pixel in yields 1 pixel out (unless the pixel is culled out of a render).

The compute shader pipeline uses one shader:

- Compute shaders - Operate on any kind of data. The output is independent of the number of threads.

Shaders are local functions and follow C style function rules. When an effect is compiled, each shader is compiled and a pointer to each shader function is stored internally. An ID3D11Effect interface is returned when compilation is successful. At this point the compiled effect is in an intermediate format.

To find out more information about the compiled shaders, you will need to use shader reflection. This is essentially like asking the runtime to decompile the shaders, and return information back to you about the shader code.

```
struct VS_OUTPUT
{
    float4 Position    : SV_POSITION; // vertex position
    float4 Diffuse     : COLOR0;      // vertex diffuse color
    float2 TextureUV   : TEXCOORD0;   // vertex texture coords
};

VS_OUTPUT RenderSceneVS( float4 vPos : POSITION,
                        float3 vNormal : NORMAL,
                        float2 vTexCoord0 : TEXCOORD,
                        uniform int nNumLights,
                        uniform bool bTexture,
                        uniform bool bAnimate )
{
    VS_OUTPUT Output;
    float3 vNormalWorldSpace;

    ....

    return Output;
}

struct PS_OUTPUT
{
    float4 RGBColor : SV_Target; // Pixel color
};

PS_OUTPUT RenderScenePS( VS_OUTPUT In,
                        uniform bool bTexture )
{
    PS_OUTPUT Output;

    if( bTexture )
        Output.RGBColor = g_MeshTexture.Sample(MeshTextureSampler,
```

```
In.TextureUV) * In.Diffuse;  
....  
return Output;  
}
```

The syntax for effect shaders is more fully detailed in [Effect Function Syntax \(Direct3D 11\)](#).

## Groups, Techniques, and Passes

A group is a collection of techniques. A technique is a collection of rendering passes (there must be at least one pass). Each effect pass (which is similar in scope to a single pass in a render loop) defines the shader state and any other pipeline state necessary to render geometry.

Groups are optional. There is a single, unnamed group which encompasses all global techniques. All other groups must be named.

Here is an example of one technique (which includes one pass) from BasicHLSL10.fx.

```
technique10 RenderSceneWithTexture1Light  
{  
    pass P0  
    {  
        SetVertexShader( CompileShader( vs_4_0, RenderSceneVS( 1, true, true  
 ) ) );  
        SetGeometryShader( NULL );  
        SetPixelShader( CompileShader( ps_4_0, RenderScenePS( true ) ) );  
    }  
}  
  
fxgroup g0  
{  
    technique11 RunComputeShader  
    {  
        pass P0  
        {  
            SetComputeShader( CompileShader( cs_5_0, CS() ) );  
        }  
    }  
}
```

The syntax for effect shaders is more fully detailed in [Effect Technique Syntax \(Direct3D 11\)](#).

# Related topics

[Effects \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Effect System Interfaces (Direct3D 11)

Article • 08/23/2019

The effect system defines several interfaces for managing effect state. There are two types of interfaces: those used by the runtime to render an effect and reflection interfaces for getting and setting effect variables.

- [Effect Runtime Interfaces](#)
- [Effect Reflection Interfaces](#)

## Effect Runtime Interfaces

Use runtime interfaces to render an effect.

Runtime Interfaces	Description
<a href="#">ID3DX11Effect</a>	Collection of one or more groups and techniques for rendering.
<a href="#">ID3DX11EffectPass</a>	A collection of state assignments.
<a href="#">ID3DX11EffectTechnique</a>	A collection of one or more passes.
<a href="#">ID3DX11EffectGroup</a>	A collection of one or more techniques.

## Effect Reflection Interfaces

Reflection is implemented in the effect system to support reading (and writing) effect state. There are multiple ways to access effect variables.

## Setting Groups of Effect State

Use these interfaces to get and set a group of state.

Reflection Interfaces	Description
<a href="#">ID3DX11EffectBlendVariable</a>	Get and set blend state.
<a href="#">ID3DX11EffectDepthStencilVariable</a>	Get and set depth-stencil state.
<a href="#">ID3DX11EffectRasterizerVariable</a>	Get and set rasterizer state.

Reflection Interfaces	Description
<a href="#">ID3DX11EffectSamplerVariable</a>	Get and set sampler state.

## Setting Effect Resources

Use these interfaces to get and set resources.

Reflection Interfaces	Description
<a href="#">ID3DX11EffectConstantBuffer</a>	Access data in a texture buffer or constant buffer.
<a href="#">ID3DX11EffectDepthStencilViewVariable</a>	Access data in a depth-stencil resource.
<a href="#">ID3DX11EffectRenderTargetViewVariable</a>	Access data in a render target.
<a href="#">ID3DX11EffectShaderResourceVariable</a>	Access data in a shader resource.
<a href="#">ID3DX11EffectUnorderedAccessViewVariable</a>	Access data in an unordered access view.

## Setting Other Effect Variables

Use these interfaces to get and set state by the variable type.

Reflection Interfaces	Description
<a href="#">ID3DX11EffectClassInstanceVariable</a>	Get a class instance.
<a href="#">ID3DX11EffectInterfaceVariable</a>	Get and set an interface.
<a href="#">ID3DX11EffectMatrixVariable</a>	Get and set a matrix.
<a href="#">ID3DX11EffectScalarVariable</a>	Get and set a scalar.
<a href="#">ID3DX11EffectShaderVariable</a>	Get a shader variable.
<a href="#">ID3DX11EffectStringVariable</a>	Get and set a string.
<a href="#">ID3DX11EffectType</a>	Get a variable type.
<a href="#">ID3DX11EffectVectorVariable</a>	Get and set a vector.

All reflection interfaces derive from [ID3DX11EffectVariable](#).

## Related topics

[Effects \(Direct3D 11\)](#)

[Programming Guide for Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Specializing Interfaces (Direct3D 11)

Article • 08/23/2019

[ID3DX11EffectVariable](#) has a number of methods for casting the interface into the particular type of interface you need. The methods are of the form *AsType* and include a method for each type of effect variable (such as AsBlend, AsConstantBuffer etc..)

For example, suppose you have an effect with two global variables: time and a world transform.

```
float      g_fTime;
float4x4   g_mWorld;
```

Here is an example that gets these variables:

```
ID3DX11EffectVariable* g_pVariable;
ID3DX11EffectMatrixVariable* g_pmWorld;
ID3DX11EffectScalarVariable* g_pfTime;

g_pVariable = g_pEffect11->GetVariableByName("g_mWorld");
g_pmWorld = g_pVariable->AsMatrix();
g_pVariable = g_pEffect11->GetVariableByName("g_fTime");
g_pfTime = g_pVariable->AsScalar();
```

By specializing the interfaces, you could reduce the code to a single call.

```
g_pmWorld = (g_pEffect11->GetVariableByName("g_mWorld"))->AsMatrix();
g_pfTime = (g_pEffect11->GetVariableByName("g_fTime"))->AsScalar();
```

Interfaces that inherit from [ID3DX11EffectVariable](#) also have these methods, but they have been designed to return invalid objects; only calls from [ID3DX11EffectVariable](#) return valid objects. Applications can test the returned object to see if it is valid by calling [ID3DX11EffectVariable::IsValid](#).

## Related topics

[Effects \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Interfaces and Classes in Effects

Article • 08/19/2020

There are many ways to use classes and interfaces in Effects 11. For interface and class syntax, see [Interfaces and Classes](#).

The following sections detail how to specify class instances to a shader which uses interfaces. We will use the following interface and classes in the examples:

```
interface IColor
{
    float4 GetColor();
};

class CRed : IColor
{
    float4 GetColor() { return float4(1,0,0,1); }
};
class CGreen : IColor
{
    float4 GetColor() { return float4(0,1,0,1); }
};

CRed pRed;
CGreen pGreen;
IColor pIColor;
IColor pIColor2 = pRed;
```

Note that interface instances can be initialized to class instances. Arrays of class and interface instances are also supported and they can be initialized as in the following example:

```
CRed pRedArray[2];
IColor pIColor3 = pRedArray[1];
IColor pIColorArray[2] = {pRed, pGreen};
IColor pIColorArray2[2] = pRedArray;
```

## Uniform Interface Parameters

Just like other uniform data types, uniform interface parameters must be specified in the `CompileShader` call. Interface parameters can be assigned to global interface instances

or global class instances. When assigned to a global interface instance, the shader will have a dependency on the interface instance, which means that it must be set to a class instance. When assigned to global class instances, the compiler specializes the shader (as with other uniform data types) to use that class. This is important for two scenarios:

1. Shaders with a `4_x` target can use interface parameters if these parameters are uniform and assigned to global class instances (so no dynamic linkage is used).
2. Users can decide to have many compiled, specialized shaders with no dynamic linkage or few compiled shaders with dynamic linkage.

```
float4 PSUniform( uniform IColor color ) : SV_Target
{
    return color;
}

technique11
{
    pass
    {
        SetPixelShader( CompileShader( ps_4_0, PSUniform(pRed) ) );
    }
    pass
    {
        SetPixelShader( CompileShader( ps_5_0, PSUniform(pIColor2) ) );
    }
}
```

If `pIColor2` remains unchanged through the API, then the previous two passes are functionally equivalent, but the first uses a `ps_4_0` static shader while the second uses a `ps_5_0` shader with dynamic linkage. If `pIColor2` is changed through the effects API (see Setting Class Instances below), then the behavior of the pixel shader in the second pass may change.

## Non-Uniform Interface Parameters

Non-uniform interface parameters create interface dependencies for the shaders. When applying a shader with interface parameters, these parameters must be assigned in with the `BindInterfaces` call. Global interface instances and global class instances can be specified in the `BindInterfaces` call.

```
float4 PSAbstract( IColor color ) : SV_Target
{
```

```

    return color;
}

PixelShader pPSAbstract = CompileShader( ps_5_0, PSAbstract(pRed) );

technique11
{
    pass
    {
        SetPixelShader( BindInterfaces( pPSAbstract, pRed ) );
    }
    pass
    {
        SetPixelShader( BindInterfaces( pPSAbstract, pIColor2 ) );
    }
}

```

If pIColor2 remains unchanged through the API, then the previous two passes are functionally equivalent and both use dynamic linkage. If pIColor2 is changed through the effects API (see Setting Class Instances below), then the behavior of the pixel shader in the second pass may change.

## Setting Class Instances

When setting a shader with dynamic shader linkage to the Direct3D 11 device, class instances must also be specified. It is an error to set such a shader with a **NULL** class instance. Therefore, all interface instances which a shader references must have an associated class instance.

The following example shows how to get a class instance variable from an effect and set it to an interface variable:

```

ID3DX11EffectPass* pPass = pEffect->GetTechniqueByIndex(0)-
>GetPassByIndex(1);

ID3DX11EffectInterfaceVariable* pIface = pEffect->GetVariableByName(
    "pIColor2" )->AsInterface();
ID3DX11EffectClassInstanceVariable* pCI = pEffect->GetVariableByName(
    "pGreen" )->AsClassInstance();
pIface->SetClassInstance( pCI );
pPass->Apply( 0, pDeviceContext );

// Apply the same pass with a different class instance
pCI = pEffect->GetVariableByName( "pRedArray" )->GetElement(1)-
>AsClassInstance();
pIface->SetClassInstance( pCI );
pPass->Apply( 0, pDeviceContext );

```

## Related topics

[Effects \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Rendering an Effect (Direct3D 11)

Article • 06/16/2021

An effect can be used to store information, or to render using a group of state. Each technique specifies some set of vertex shaders, hull shaders, domain shaders, geometry shaders, pixel shaders, compute shaders, shader state, sampler and texture state, unordered access view state and other pipeline state. So once you have organized state into an effect, you can encapsulate the rendering effect that results from that state by creating and rendering the effect.

There are a few steps for preparing an effect for rendering. The first is compiling, which checks the HLSL like code against the HLSL language syntax and the effect framework rules. You can compile an effect from your application using API calls or you can compile an effect offline using the effect compiler utility [fxc.exe](#). Once your effect has successfully compiled, create it by calling a different (but very similar) set of APIs.

After the effect is created, there are two remaining steps for using it. First, you must initialize the effect state values (the values of the effect variables) using a number of methods for setting state if they are not initialized in HLSL. For some variables this can be done once when an effect is created; others must be updated each time your application calls the render loop. Once the effect variables are set, you tell the runtime to render the effect by applying a technique. These topics are all discussed in further detail below.

Naturally there are performance considerations for using effects. These considerations are largely the same if you are not using an effect. Things like minimizing the amount of state changes and organizing the variables by frequency of update. These tactics are used to minimize the amount of data that needs to be sent from the CPU to the GPU, and therefore minimize potential synchronization problems.

## In this section

Topic	Description
<a href="#">Compile an Effect</a>	After an effect has been authored, the next step is to compile the code to check for syntax problems.
<a href="#">Create an Effect</a>	An effect is created by loading the compiled effect bytecode into the effects framework. Unlike Effects 10, the effect must be compiled before creating the effect. Effects loaded into memory can be created by calling <a href="#">D3DX11CreateEffectFromMemory</a> .

Topic	Description
Set Effect State	Some effect constants only need to be initialized. Once initialized, the effect state is set to the device for the entire render loop. Other variables need to be updated each time the render loop is called. The basic code for setting effect variables is shown below, for each of the types of variables.
Apply a Technique	With the constants, textures, and shader state declared and initialized, the only thing left to do is to set the effect state in the device.

## Related topics

[Effects \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Compile an Effect (Direct3D 11)

Article • 08/19/2020

After an effect has been authored, the next step is to compile the code to check for syntax problems.

You do so by calling one of the compile APIs ([D3DX11CompileFromFile](#), [D3DX11CompileFromMemory](#), or [D3DX11CompileFromResource](#)). These APIs invoke the effect compiler fxc.exe, which compiles HLSL code. This is why the syntax for code in an effect looks very much like HLSL code. (There are a few exceptions that will be handled later). The effect compiler/hlsl compiler, fxc.exe, is available in the SDK in the utilities folder so that shaders (or effects) can be compiled offline if you choose. See the documentation for running the compiler from the command line.

- [Example](#)
- [Includes](#)
- [Searching for Include Files](#)
- [Macros](#)
- [HLSL Shader Flags](#)
- [FX Flags](#)
- [Checking Errors](#)
- [Related topics](#)

## Example

Here's an example of compiling an effect file.

```
WCHAR str[MAX_PATH];
DXUTFindDXSDKMediaFileCch( str, MAX_PATH, L"BasicHLSL10.fx" );

hr = D3DX11CompileFromFile( str, NULL, NULL, pFunctionName, pProfile,
D3D10_SHADER_ENABLE_STRICTNESS, NULL, NULL, &pBlob, &pErrorBlob, NULL );
```

## Includes

One parameter of the compile APIs is an include interface. Generate one of these if you want to include a customized behavior when the compiler reads an include file. The compiler executes this custom behavior each time it creates or compiles an effect (that uses the include pointer). To implement customized include behavior, derive a class from

the [ID3DInclude](#) interface. This provides your class with two methods: [Open](#) and [Close](#). Implement the custom behavior in these methods.

## Searching for Include Files

The pointer that the compiler passes in the *pParentData* parameter to your include handler's [Open](#) method might not point to the container that includes the #include file that the compiler needs to compile your shader code. That is, the compiler might pass **NULL** in *pParentData*. Therefore, we recommend that your include handler search its own list of include locations for content. Your include handler can dynamically add new include locations as it receives those locations in calls to its [Open](#) method.

In the following example, suppose that the shader code's include files are both stored in the *somewhereelse* directory. When the compiler calls the include handler's [Open](#) method to open and read the contents of *somewhereelse\foo.h*, the include handler can save the location of the **somewhereelse** directory. Later, when the compiler calls the include handler's [Open](#) method to open and read the contents of *bar.h*, the include handler can automatically search in the *somewhereelse* directory for *bar.h*.

```
Main.hlsl:  
#include "somewhereelse\foo.h"  
  
Foo.h:  
#include "bar.h"
```

## Macros

Effect compilation can also take a pointer to macros that are defined elsewhere. For example, suppose you want to modify the effect in BasicHLSL10, to use two macros: zero and one. The effect code that uses the two macros is shown here.

```
if( bAnimate )  
    vAnimatedPos += float4(vNormal, zero) *  
        (sin(g_fTime+5.5)+0.5)*5;  
  
Output.Diffuse.a = one;
```

Here is the declaration for the two macros.

```
D3D10_SHADER_MACRO Shader_Macros[3] = { "zero", "0", "one", "1.0f", NULL,  
NULL };
```

The macros are a NULL-terminated array of macros; where each macro is defined by using a [D3D10\\_SHADER\\_MACRO](#) struct.

Modify the compile effect call to take a pointer to the macros.

```
D3DX11CompileFromFile( str, Shader_Macros, NULL, pFunctionName,  
pProfile, D3D10_SHADER_ENABLE_STRICTNESS, NULL,  
NULL, &pBlob, &pErrorBlob, NULL );
```

## HLSL Shader Flags

Shader flags specify shader constraints to the HLSL compiler. These flags affect the code generated by the shader compiler in the following ways:

- Optimize the code size.
- Including debug information, which prevents flow control.
- Affects the compile target and whether a shader can run on legacy hardware.

These flags can be logically combined if you have not specified two conflicting characteristics. For a listing of the flags see [D3D10\\_SHADER Constants](#).

## FX Flags

Use these flags when you create an effect to define either compilation behavior or runtime effect behavior. For a listing of the flags see [D3D10\\_EFFECT Constants](#).

## Checking Errors

If during compilation an error occurs, the API returns an interface that contains the errors from the effect compiler. This interface is called [ID3DBlob](#). It is not directly readable; however, by returning a pointer to the buffer that contains the data (which is a string), you can see any compilation errors.

This example contains an error in the BasicHLSL.fx, the first variable declaration occurs twice.

```

//-----
// Global variables
//-----
float4 g_MaterialAmbientColor;      // Material's ambient color

// Declare the same variable twice
float4 g_MaterialAmbientColor;      // Material's ambient color

```

This error causes the compiler to return the following error, as shown in the following screen shot of the Watch window in Microsoft Visual Studio.

Watch 1	
Name	Value
+ (char*)l_pError	0x01997fb8 "BasicHLSL10.fx(18): error X3003: redefinition of 'g_MaterialAmbientColor'"

Because the compiler returns the error in a LPVOID pointer, cast it to a character string in the Watch window.

Here is the code that returns the error from the failed compile.

```

// Read the D3DX effect file
WCHAR str[MAX_PATH];
ID3DBlob* l_pBlob_Effect = NULL;
ID3DBlob* l_pBlob_Errors = NULL;
hr = DXUTFindDXSDKMediaFileCch( str, MAX_PATH, L"BasicHLSL10.fx" );
hr = D3DX11CompileFromFile( str, NULL, NULL, pFunctionName,
                           pProfile, D3D10_SHADER_ENABLE_STRICTNESS, NULL,
                           NULL, &pBlob, &pErrorBlob, NULL );

LPVOID l_pError = NULL;
if( pErrorBlob )
{
    l_pError = pErrorBlob->GetBufferPointer();
    // then cast to a char* to see it in the locals window
}

```

## Related topics

[Rendering an Effect \(Direct3D 11\)](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Create an Effect (Direct3D 11)

Article • 08/23/2019

An effect is created by loading the compiled effect bytecode into the effects framework. Unlike Effects 10, the effect must be compiled before creating the effect. Effects loaded into memory can be created by calling [D3DX11CreateEffectFromMemory](#).

## Related topics

[Rendering an Effect \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# Set Effect State (Direct3D 11)

Article • 06/18/2021

Some effect constants only need to be initialized. Once initialized, the effect state is set to the device for the entire render loop. Other variables need to be updated each time the render loop is called. The basic code for setting effect variables is shown below, for each of the types of variables.

An effect encapsulates all of the render state required to do a rendering pass. In terms of the API, there are three types of state encapsulated in an effect.

- Constant State
- Shader State
- Texture State

## Constant State

First, declare variables in an effect using HLSL data types.

```
//-----
// Global variables
//-----

float4 g_MaterialAmbientColor;           // Material's ambient color
float4 g_MaterialDiffuseColor;           // Material's diffuse color
int g_nNumLights;

float3 g_LightDir[3];                    // Light's direction in world space
float4 g_LightDiffuse[3];                // Light's diffuse color
float4 g_LightAmbient;                  // Light's ambient color

Texture2D g_MeshTexture;                // Color texture for mesh

float    g_fTime;                      // App's time in seconds
float4x4 g_mWorld;                     // World matrix for object
float4x4 g_mWorldViewProjection;        // World * View * Projection matrix
```

Second, declare variables in the application that can be set by the application, and will then update the effect variables.

```

D3DXMATRIX mWorldViewProjection;
D3DXVECTOR3 vLightDir[MAX_LIGHTS];
D3DXVECTOR4 vLightDiffuse[MAX_LIGHTS];
D3DXMATRIX mWorld;
D3DXMATRIX mView;
D3DXMATRIX mProj;

// Get the projection and view matrix from the camera class
mWorld = g_mCenterMesh * *g_Camera.GetWorldMatrix();
mProj = *g_Camera.GetProjMatrix();
mView = *g_Camera.GetViewMatrix();

OnD3D11CreateDevice()
{
    ...
    g_pLightDir = g_pEffect11->GetVariableByName( "g_LightDir" )-
>AsVector();
    g_pLightDiffuse = g_pEffect11->GetVariableByName( "g_LightDiffuse" )-
>AsVector();
    g_pmWorldViewProjection = g_pEffect11->GetVariableByName(
        "g_mWorldViewProjection" )->AsMatrix();
    g_pmWorld = g_pEffect11->GetVariableByName( "g_mWorld" )->AsMatrix();
    g_pfTime = g_pEffect11->GetVariableByName( "g_fTime" )->AsScalar();
    g_pMaterialAmbientColor = g_pEffect11-
>GetVariableByName("g_MaterialAmbientColor")->AsVector();
    g_pMaterialDiffuseColor = g_pEffect11->GetVariableByName(
        "g_MaterialDiffuseColor" )->AsVector();
    g_pnNumLights = g_pEffect11->GetVariableByName( "g_nNumLights" )-
>AsScalar();
}

```

Third, use the update methods to set the value of the variables in the application in the effect variables.

```

OnD3D11FrameRender()
{
    ...
    g_pLightDir->SetRawValue( vLightDir, 0, sizeof(D3DXVECTOR3)*MAX_LIGHTS );
    g_pLightDiffuse->SetFloatVectorArray( (float*)vLightDiffuse, 0,
MAX_LIGHTS );
    g_pmWorldViewProjection->SetMatrix( (float*)&mWorldViewProjection );
    g_pmWorld->SetMatrix( (float*)&mWorld );
    g_pfTime->SetFloat( (float)fTime );
    g_pnNumLights->SetInt( g_nNumActiveLights );
}

```

## Two Ways to Get the State in an Effect Variable

There are two ways to get the state contained in an effect variable. Given an effect that has been loaded into memory.

One way is to get the sampler state from an [ID3DX11EffectVariable](#) that has been cast as a sampler interface.

```
D3D11_SAMPLER_DESC sampler_desc;
ID3D11EffectSamplerVariable* l_pD3D11EffectVariable = NULL;
if( g_pEffect11 )
{
    l_pD3D11EffectVariable = g_pEffect11->GetVariableByName(
    "MeshTextureSampler" )->AsSampler();
    if( l_pD3D11EffectVariable->IsValid() )
        hr = (l_pD3D11EffectVariable->GetBackingStore( 0,
            &sampler_desc );
}
```

The other way is to get the sampler state from an [ID3D11SamplerState](#).

```
ID3D11SamplerState* l_ppSamplerState = NULL;
D3D11_SAMPLER_DESC sampler_desc;
ID3D11EffectSamplerVariable* l_pD3D11EffectVariable = NULL;
if( g_pEffect11 )
{
    l_pD3D11EffectVariable = g_pEffect11->GetVariableByName(
    "MeshTextureSampler" )->AsSampler();
    if( l_pD3D11EffectVariable->IsValid )
    {
        hr = l_pD3D11EffectVariable->GetSampler( 0,
            &l_ppSamplerState );
        if( l_ppSamplerState )
            l_ppSamplerState->GetDesc( &sampler_desc );
    }
}
```

## Shader State

Shader state is declared and assigned in an effect technique, within a pass.

```

VertexShader vsRenderScene = CompileShader( vs_4_0, RenderSceneVS( 1, true,
true );
technique10 RenderSceneWithTexture1Light
{
    pass P0
    {
        SetVertexShader( vsRenderScene );
        SetGeometryShader( NULL );
        SetPixelShader( CompileShader( ps_4_0, RenderScenePS( true ) ) );
    }
}

```

This works just like it would if you were not using an effect. There are three calls, one for each type of shader (vertex, geometry, and pixel). The first one, SetVertexShader, calls [ID3D11DeviceContext::VSSetShader](#). CompileShader is a special effect function that takes the shader profile(vs\_4\_0) and the name of the vertex shader function (RenderVS). In other words, each of these CompileShader calls compiles their associated shader function and returns a pointer to the compiled shader.

Note that not all shader state must be set. This pass does not include any SetHullShader or SetDomainShader calls, meaning that the currently bound hull and domain shaders will be unchanged.

## Texture State

Texture state is a little more complex than setting a variable, because texture data is not simply read like a variable, it is sampled from a texture. Therefore, you must define the texture variable (just like a normal variable except it uses a texture type) and you must define the sampling conditions. Here is an example of a texture variable declaration and the corresponding sampling state declaration.

```

Texture2D g_MeshTexture;           // Color texture for mesh

SamplerState MeshTextureSampler
{
    Filter = MIN_MAG_MIP_LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};

```

Here is an example of setting a texture from an application. In this example, the texture is stored in the mesh data, that was loaded when the effect was created.

The first step is getting a pointer to the texture from the effect (from the mesh).

```
ID3D11EffectShaderResourceVariable* g_ptxDiffuse = NULL;  
  
// Obtain variables  
g_ptxDiffuse = g_pEffect11->GetVariableByName( "g_MeshTexture" )->AsShaderResource();
```

The second step is specifying a view for accessing the texture. The view defines a general way to access the data from the texture resource.

```
OnD3D11FrameRender()  
{  
    ID3D11ShaderResourceView* pDiffuseRV = NULL;  
  
    ...  
    pDiffuseRV = g_Mesh11.GetMaterial(pSubset->MaterialID)->pDiffuseRV11;  
    g_ptxDiffuse->SetResource( pDiffuseRV );  
    ...  
}
```

From the application perspective, unordered access views are handled similarly to shader resource views. However, in the effect pixel shader and compute shader functions, unordered access view data is read from/written to directly. You cannot sample from an unordered access view.

For more information about viewing resources, see [Resources](#).

## Related topics

[Rendering an Effect \(Direct3D 11\)](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Apply a Technique (Direct3D 11)

Article • 06/30/2021

With the constants, textures, and shader state declared and initialized, the only thing left to do is to set the effect state in the device.

## Set Non-Shader State in the Device

Some pipeline state is not set by an effect. For example clearing a render target prepares the render target for data. Before setting the effect state in the device, here is an example of clearing output buffers.

```
// Clear the render target and depth stencil
float ClearColor[4] = { 0.0f, 0.25f, 0.25f, 0.55f };
ID3D11RenderTargetView* pRTV = DXUTGetD3D11RenderTargetView();
pD3DDevice->ClearRenderTargetView( pRTV, ClearColor );
ID3D11DepthStencilView* pDSV = DXUTGetD3D11DepthStencilView();
pD3DDevice->ClearDepthStencilView( pDSV, D3D11_CLEAR_DEPTH, 1.0, 0 );
```

## Set Effect State in the Device

Setting effect state is done by applying the effect state within the render loop. This is done from the outside in. That is, select a technique, and then set the state for each of the passes (depending on your desired result).

```
D3D11_TECHNIQUE_DESC techDesc;
pRenderTechnique->GetDesc( &techDesc );
for( UINT p = 0; p < techDesc.Passes; ++p )
{
    ...
    ...
    pRenderTechnique->GetPassByIndex( p )->Apply(0);
    pd3dDevice->DrawIndexed( (UINT)pSubset->IndexCount, 0,
                            (UINT)pSubset->VertexStart );
}
```

An effect doesn't render anything, it simply sets effect state to the device. The rendering code is called after the effect state updates device state. In this example, the

DrawIndexed call performs the rendering.

## Related topics

[Rendering an Effect \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?



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

# Cloning an Effect

Article • 08/23/2019

Cloning an effect creates a second, almost identical copy of the effect. See the following single qualifier for an explanation of why it is not exact. A second copy of an effect is useful when one wants to use the effects framework on multiple threads, since the effect runtime is not thread safe to maintain high performance.

Since device contexts are also non-thread-safe, different threads should pass different device contexts to the ID3DX11EffectPass::Apply method.

An effect can be cloned with the following syntax:

```
ID3DX11Effect* pClonedEffect = NULL;  
UINT Flags = D3DX11_EFFECT_CLONE_FORCE_NONSINGLE;  
HRESULT hr = pEffect->CloneEffect( Flags, &pClonedEffect );
```

In the above example, the cloned copy will encapsulate the same state as the original effect, regardless of what state the original effect is in. In particular:

1. If pEffect is optimized, pCloned Effect is optimized
2. If pEffect has some user-managed variables, pCloned Effect will have the same user-managed variables (see the single description below)
3. Any pending variable updates (until an Apply call updates device state) in pEffect will be pending in pClonedEffect

The following Direct3D 11 device objects are immutable or never updated by the effects framework, so the cloned effect will point to the same objects as the original effect:

1. State block objects (ID3D11BlendState, ID3D11RasterizerState, ID3D11DepthStencilState, ID3D11SamplerState)
2. Shaders
3. Class instances
4. Textures (not including texture buffers)
5. Unordered access views

The following Direct3D 11 device objects are both immutable and modified by the effect runtime (unless user-managed or single in a cloned effect); new copies of these objects are created when non-single:

1. Constant buffers

## Single Constant Buffers and Texture Buffers

Note that this discussion applies to both constant buffers and textures, but constant buffers are assumed for ease of reading.

There may be cases where a constant buffer is only updated by one thread, but device state set by cloned effects will use this data. For example, the main effect may update the world and view matrices which are referenced from shaders in cloned effects which do not change the world and view matrices. In these cases, the cloned effects need to reference the current constant buffer instead of recreating one.

There are two ways to achieve this desired result:

1. Use `ID3DX11EffectConstantBuffer::SetConstantBuffer` on the cloned effect to make it user-managed
2. Mark the constant buffer as "single" in the HLSL code, forcing the effect runtime to treat it as user-managed after cloning

There are two differences between the two methods above. First, in method 1, a new `ID3D11Buffer` will be created and used before `SetConstantBuffer` is called. Also, after calling `UndoSetConstantBuffer` in the cloned effect, the variable in method 1 will point to the newly-created buffer (which effects will update on `Apply`) while the variable in method 2 will continue to point to the original buffer (not updating it on `Apply`).

See the following example in HLSL:

```
cbuffer ObjectData
{
    float4 Position;
};

single cbuffer ViewData
{
    float4x4 ViewMatrix;
};
```

While cloning, the cloned effect will create a new `ID3D11Buffer` for `ObjectData`, and fill its contents on `Apply`, but reference the original `ID3D11Buffer` for `ViewData`. The `single` qualifier can be ignored in the cloning process by setting the `D3DX11_EFFECT_CLONE_FORCE_NONSINGLE` flag.

# Related topics

[Effects \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Stream Out Syntax

Article • 08/23/2019

A geometry shader with stream out is declared with a particular syntax. This topic describes the syntax. In the effect runtime, this syntax will be converted to a call to [ID3D11Device::CreateGeometryShaderWithStreamOutput](#).

## Construct Syntax

```
[ StreamingShaderVar = ] ConstructGSWithSO( ShaderVar, "OutputDecl0" )
```

Name	Description
StreamingShaderVar	Optional. An ASCII string that uniquely identifies the name of a geometry shader variable with stream out. This is optional because ConstructGSWithSO can be placed directly in a SetGeometryShader or BindInterfaces call.
ShaderVar	A geometry shader or vertex shader variable.
OutputDecl0	A string defining which shader outputs in stream 0 are streamed out. See below for syntax.

This is the syntax was defined in fx\_4\_0 files. Note that in gs\_4\_0 and vs\_x shaders, there is only one stream of data. The resulting shader will output one stream to both the stream out unit and the rasterizer unit.

```
[ StreamingShaderVar = ] ConstructGSWithSO( ShaderVar, "OutputDecl0",
"OutputDecl1", "OutputDecl2",
"OutputDecl3", RasterizedStream )
```

Name	Description
StreamingShaderVar	Optional. An ASCII string that uniquely identifies the name of a geometry shader variable with stream out. This is optional because ConstructGSWithSO can be placed directly in a SetGeometryShader or BindInterfaces call.

Name	Description
ShaderVar	A geometry shader or vertex shader variable.
OutputDecl0	A string defining which shader outputs in stream 0 are streamed out. See below for syntax.
OutputDecl1	A string defining which shader outputs in stream 1 are streamed out. See below for syntax.
OutputDecl2	A string defining which shader outputs in stream 2 are streamed out. See below for syntax.
OutputDecl3	A string defining which shader outputs in stream 3 are streamed out. See below for syntax.
RasterizedStream	An integer specifying which stream will be sent to the rasterizer.

Note that gs\_5\_0 shaders can define up to four streams of data. The resulting shader will output one stream to the stream out unit for each non-**NULL** output declaration and one stream the rasterizer unit.

## Stream Out Declaration Syntax

```
" [ Buffer: ] Semantic[ SemanticIndex ] [ .Mask ]; [ ... ; ] ... [ ... ; ]"
```

Name	Description
Buffer	Optional. An integer, $0 \leq \text{Buffer} < 4$ , specifying which stream out buffer the value will go to.
Semantic	A string, along with SemanticIndex, specifying which value to output.
SemanticIndex	Optional. The index associated with Semantic.
Mask	Optional. A component mask, indicating which components of the value to output.

There is one special Semantic, labeled "\$SKIP" which indicates an empty semantic, leaving the corresponding memory in the stream out buffer untouched. The \$SKIP semantic cannot have a SemanticIndex, but can have a Mask.

The entire stream out declaration can be **NULL**.

## Example

```
struct GSOutput
{
    int4 Pos : Position;
    int4 Color : Color;
    int4 Texcoord : Texcoord;
};

[maxvertexcount(1)]
void gsBase (inout PointStream<GSOutput> OutputStream, inout
PointStream<GSOutput> OutputStream1)
{
    GSOutput output;
    output.Pos = int4(1,2,3,4);
    output.Color = int4(5,6,7,8);
    output.Texcoord = int4(9,10,11,12);
    OutputStream.Append(output);

    output.Pos = int4(1,2,3,4);
    output.Color = int4(5,6,7,8);
    output.Texcoord = int4(9,10,11,12);
    OutputStream1.Append(output);
}

GeometryShader pGSComp = CompileShader(gs_5_0, gsBase());
GeometryShader pGSwSO = ConstructGSWithSO(pGSComp, "0:Position.xy;
1:Position.zw; 2:Color.xy",
                                         "3:Texcoord.xyzw;
3:$SKIP.x;", NULL, NULL, 1);

// The following two passes perform the same operation
technique11 SOPoints
{
    pass
    {
        SetGeometryShader(ConstructGSWithSO(pGSComp, "0:Position.xy;
1:Position.zw; 2:Color.xy",
                                         "3:Texcoord.xyzw;
3:$SKIP.x;", NULL, NULL, 1));
    }
    pass
    {
        SetGeometryShader(pGSwSO);
    }
}
```

# Related topics

[Effects \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Differences Between Effects 10 and Effects 11

Article • 08/19/2020

This topic shows the differences between Effects 10 and Effects 11.

## Device Contexts, Threading, and Cloning

The ID3D10Device interface has been split into two interfaces in Direct3D 11: ID3D11Device and ID3D11DeviceContext. You can create multiple ID3D11DeviceContexts to facilitate concurrent execution on multiple threads. Effects 11 extends this concept to the Effects framework.

The Effects 11 runtime is single-threaded. For this reason, you should not use a single ID3DX11Effect instance with multiple threads concurrently.

To use the Effects 11 runtime on multiple instances, you must create separate ID3DX11Effect instances. Because the ID3D11DeviceContext is also single-threaded, you should pass different ID3D11DeviceContext instances to each effect instance on Apply. You can use these separate device contexts to create command lists so that the rendering thread can apply them on the immediate device context.

The easiest way to create multiple effects that encapsulate the same functionality, for use on multiple threads, is to create one effect and then make cloned copies. Cloning has the following advantages over creating multiple copies from scratch:

1. The cloning routine is faster than the creation routine.
2. Cloned effects share created shaders, state blocks, and class instances (so they don't have to be recreated).
3. Cloned effects can share constant buffers.
4. Cloned effects begin with state that matches the current effect (variable values, whether or not it has been optimized).

See [Cloning an Effect](#) for more information.

## Effect Pools and Groups

By far the most prevalent use of effect pools in Direct3D 10 was for grouping materials. Effect Pools have been removed from Effects 11 and groups have been added, which is a more efficient method of grouping materials.

An effect group is simply a set of techniques. See [Effect Group Syntax \(Direct3D 11\)](#) for more information.

Consider the following effect hierarchy with four child effects and one effect pool:

```
// Effect Pool
cbuffer A { ... }
PixelShader pPSGrass;
PixelShader pPSWater;

// Effect Child 1
#include "EffectPool.fx"
technique10 GrassMaterialLowSpec { ... }

// Effect Child 2
#include "EffectPool.fx"
technique10 GrassMaterialHighSpec { ... }

// Effect Child 3
#include "EffectPool.fx"
technique10 WaterMaterialLowSpec { ... }

// Effect Child 4
#include "EffectPool.fx"
technique10 WaterMaterialHighSpec { ... }
```

You can achieve the same functionality in Effects 11 by using groups:

```
cbuffer A { ... }
PixelShader pPSGrass;
PixelShader pPSWater;

fxgroup GrassMaterial
{
    technique10 LowSpec { ... }
    technique10 HighSpec { ... }
}

fxgroup WaterMaterial
{
    technique10 LowSpec { ... }
    technique10 HighSpec { ... }
}
```

## New Shader Stages

There are three new shader stages in Direct3D 11: the hull shader, domain shader, and compute shader. Effects 11 handles these in a similar manner to vertex shaders, geometry shaders, and pixel shaders.

Three new variable types have been added to Effects 11:

- HullShader
- DomainShader
- ComputeShader

If you use these shaders in a technique, you must label that technique "technique11", and not "technique10". The compute shader cannot be set in the same pass as any other graphics state (other shaders, state blocks, or render targets).

## New Texture Types

Direct3D 11 supports the following texture types:

- AppendStructuredBuffer
- ByteAddressBuffer
- ConsumeStructuredBuffer
- StructuredBuffer

## Unordered Access Views

Effects 11 supports getting and setting the new unordered access view types. This works in a similar manner as textures.

Consider this Effects HLSL example:

```
RWTexture1D<float> myUAV;
```

You can set this variable in C++ as follows:

```
ID3D11UnorderedAccessView* pUAVTexture1D = NULL;  
ID3DX11EffectUnorderedAccessViewVariable* pUAVVar;  
pUAVVar = pEffect->GetVariableByName("myUAV")->AsUnorderedAccessView();  
pUAVVar->SetUnorderedAccessView( pUAVTexture1D );
```

---

Direct3D 11 supports the following unordered access view types:

- RWBuffer
- RWByteAddressBuffer
- RWStructuredBuffer
- RWTexure1D
- RWTexure1DArray
- RWTexure2D
- RWTexure2DArray
- RWTexure3D

## Interfaces and Class Instances

For interface and class syntax, see [Interfaces and Classes](#).

For using interfaces and classes in effects, see [Interfaces and Classes in Effects](#).

## Addressable Stream Out

In Direct3D 10, geometry shaders could output one stream of data to the stream output unit and the rasterizer unit. In Direct3D11, geometry shaders can output up to four streams of data to the stream output unit, and at most one of those streams to the rasterizer unit. The `ConstructGSWithSO` intrinsic has been updated to reflect this new functionality.

See [Stream Out Syntax](#) for more information.

## Setting and Unsetting Device State

In Effects 10, you could make constant buffers and texture buffers user-managed by using the `ID3D10EffectConstantBuffer::SetConstantBuffer` and `SetTextureBuffer` functions. After you call these functions, the Effects 10 runtime no longer manages the constant buffer or texture buffer and the user must fill the data by using the `ID3D10Device` interface.

In Effects 11, you can also make the state blocks (blend state, rasterizer state, depth-stencil state, and sampler state) user-managed by using the following calls:

- [ID3DX11EffectBlendVariable::SetBlendState](#)
- [ID3DX11EffectRasterizerVariable::SetRasterizerState](#)
- [ID3DX11EffectDepthStencilVariable::SetDepthStencilState](#)

- [ID3DX11EffectSamplerVariable::SetSampler](#)

After you call these functions, the Effects 11 runtime no longer manages the state block variables, and their values will remain unchanged. Note that because state blocks are immutable, the user must set a new state block to change the values.

You can also revert constant buffers, texture buffers, and state blocks to the non-user managed state. If you unset these variables, the Effects 11 runtime will continue to update them when necessary. You can use the following calls to unset user managed variables:

- [ID3DX11EffectConstantBuffer::UndoSetConstantBuffer](#)
- [ID3DX11EffectConstantBuffer::UndoSetTextureBuffer](#)
- [ID3DX11EffectBlendVariable::UndoSetBlendState](#)
- [ID3DX11EffectRasterizerVariable::UndoSetRasterizerState](#)
- [ID3DX11EffectDepthStencilVariable::UndoSetDepthStencilState](#)
- [ID3DX11EffectSamplerVariable::UndoSetSampler](#)

## Effects Virtual Machine

The effects virtual machine, which evaluated complex expressions outside of functions, has been removed.

The following examples of complex expressions are not supported:

1. SetPixelShader( myPSArray( i \* 3 + j ) );
2. SetPixelShader( myPSArray( float)i );
3. FILTER = i + 2;

The following examples of non-complex expressions are supported:

1. SetPixelShader( myPS );
2. SetPixelShader( myPS[i] );
3. SetPixelShader( myPS[ ulIndex ] ); // ulIndex is a uint variable
4. FILTER = i;
5. FILTER = ANISOTROPIC;

These expressions could appear in state block expressions (such as FILTER) and pass expressions (such as SetPixelShader).

## Source Availability and Location

Effects 10 was distributed in D3D10.dll. Effects 11 is distributed as source, with corresponding Visual Studio solutions to compile it. When you create effects-type applications, we recommend that you include the Effects 11 source directly in those applications.

You can get Effects 11 from [Effects for Direct3D 11 Update](#).

## Related topics

[Effects \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Migrating to Direct3D 11

Article • 10/06/2021

This section provides info for migrating to Direct3D 11 from an earlier version of Direct3D.

- [Direct3D 9 to Direct3D 11](#)
- [Direct3D 10 to Direct3D 11](#)
  - [Enumerations and Defines](#)
  - [Structures](#)
  - [Interfaces](#)
  - [Other Related Technologies](#)
- [Direct3D 10.1 to Direct3D 11](#)
  - [Enumerations and Defines](#)
  - [Structures](#)
  - [Interfaces](#)
- [New Features for Direct3D 11](#)
- [New Features for DirectX 11.1](#)
- [New Features for DirectX 11.2](#)
- [Related topics](#)

## Direct3D 9 to Direct3D 11

The Direct3D 11 API builds on the infrastructural improvements made in Direct3D 10 and 10.1. Porting from Direct3D 9 to Direct3D 11 is similar to moving from Direct3D 9 to Direct3D 10. The following are the key challenges in this effort.

- Removal of all fixed function pipeline usage in favor of programmable shaders authored exclusively in HLSL (compiled via D3DCompiler instead of D3DX9).
- State management based on immutable objects rather than individual state toggles.
- Updating to comply with strict linkage requirements of vertex buffer input layouts and shader signatures.
- Associating shader resource views with all texture resources.
- Mapping all image content to a DXGI\_FORMAT, including the removal of all 16-bit color formats (5/5/5/1, 5/6/5, 4/4/4/4), removal of all 24-bit color formats (8/8/8), and strict RGB color ordering.
- Breaking up global constant state usage into several small, more efficiently updated constant buffers.

For more information about moving from Direct3D 9 to Direct3D 10, see [Direct3D 9 to Direct3D 10 Considerations](#).

## Direct3D 10 to Direct3D 11

Converting programs written to use the Direct3D 10 or 10.1 API is a straight-forward process as Direct3D 11 is an extension of the existing API. With only one minor exception (noted below - monochrome text filtering), all methods and functionality in Direct3D 10/10.1 is available in Direct3D 11. The outline below describes the differences between the two APIs to aid in updating existing code. The key differences here include:

- Rendering operations (Draw, state, etc.) are no longer part of the Device interface, but are instead part of the new DeviceContext interface along with resource Map/Unmap and device query methods.
- Direct3D 11 includes all enhancements and changes made between Direct3D 10.0 and 10.1

## Enumerations and Defines

Direct3D 10	Direct3D 11
DXGI_FORMAT	<a href="#">DXGI_FORMAT</a> Several new DXGI formats were defined.
D3D10_CREATE_DEVICE_SWITCH_TO_REF	<a href="#">D3D11_CREATE_DEVICE_SWITCH_TO_REF</a> The switch-to-ref functionality is not supported by Direct3D 11.
D3D10_DRIVER_TYPE	<a href="#">D3D_DRIVER_TYPE</a> Note that the enumeration identifiers in <a href="#">D3D_DRIVER_TYPE</a> were redefined from the identifiers in <a href="#">D3D10_DRIVER_TYPE</a> . Therefore, you should be sure to use the enumeration identifiers instead of literal numbers. <a href="#">D3D_DRIVER_TYPE</a> is defined in D3Dcommon.h.
D3D10_RESOURCE_MISC_FLAG	<a href="#">D3D11_RESOURCE_MISC_FLAG</a> Note that many of these flags were redefined, so be sure to use enumeration identifiers instead of literal numbers.
D3D10_FILTER	<a href="#">D3D11_FILTER</a> Note that text filtering D3D10_FILTER_TEXT_1BIT was removed from Direct3D 11. See DirectWrite.
D3D10_COUNTER	<a href="#">D3D11_COUNTER</a> Note that the vendor-neutral counters were removed for Direct3D 11 as they were rarely supported.

Direct3D 10	Direct3D 11
D3D10_x	D3D11_x Many enumerations and defines are the same, have larger limits, or have additional values.

## Structures

Direct3D 10	Direct3D 11
D3D10_SO_DECLARATION_ENTRY	<a href="#">D3D11_SO_DECLARATION_ENTRY</a> Adds Stream.
D3D10_BLEND_DESC	<a href="#">D3D11_BLEND_DESC</a> Note this structure changed significantly from 10 to 10.1 to provide per render target blend state
D3D10_BUFFER_DESC	<a href="#">D3D11_BUFFER_DESC</a> Adds a StructureByteStride for use with Compute Shader resources
D3D10_SHADER_RESOURCE_VIEW_DESC	<a href="#">D3D11_SHADER_RESOURCE_VIEW_DESC</a> Note this structure had additional union members added for 10.1
D3D10_DEPTH_STENCIL_VIEW_DESC	<a href="#">D3D11_DEPTH_STENCIL_VIEW_DESC</a> Has a new Flags member.
D3D10_QUERY_DATA_PIPELINE_STATISTICS	<a href="#">D3D11_QUERY_DATA_PIPELINE_STATISTICS</a> Adds several new shader stage counters.
D3D10_x	D3D11_x Many structures are identical between the two APIs.

## Interfaces

Direct3D 10	Direct3D 11
-------------	-------------

Direct3D 10	Direct3D 11
ID3D10Device	<p><a href="#">ID3D11Device</a> and <a href="#">ID3D11DeviceContext</a></p> <p>The device interface was split into these two parts. For quick porting you can make use of <a href="#">ID3D11Device::GetImmediateContext</a>.</p> <p>"ID3D10Device::GetTextFilterSize" and "SetTextFilerSize" methods no longer exist. See DirectWrite.</p> <p>Create*Shader takes an additional optional parameter for <a href="#">ID3D11ClassLinkage</a>.</p> <p>*SetShader and *GetShader take additional optional parameters for <a href="#">ID3D11ClassInstance(s)</a>.</p> <p><a href="#">CreateGeometryShaderWithStreamOutput</a> takes an array and count for multiple output stream strides.</p> <p>The limit for the NumEntries parameter of <a href="#">CreateGeometryShaderWithStreamOutput</a> has increased to D3D11_SO_STREAM_COUNT * D3D11_SO_OUTPUT_COMPONENT_COUNT in Direct3D 11.</p>
ID3D10Buffer	<a href="#">ID3D11Buffer</a>
ID3D10SwitchToRef	<a href="#">ID3D11SwitchToRef</a> Switch-to-ref functionality is not supported in Direct3D 11.
ID3D10Texture1D	<a href="#">ID3D11Texture1D</a>
ID3D10Texture2D	<a href="#">ID3D11Texture2D</a>
ID3D10Texture3D	<p><a href="#">ID3D11Texture3D</a> The Map and Unmap methods were moved to <a href="#">ID3D11DeviceContext</a>, and all Map methods use <a href="#">D3D11_MAPPED_SUBRESOURCE</a> instead of a void**.</p>
ID3D10Asynchronous	<a href="#">ID3D11Asynchronous</a> Begin, End, and GetData were moved to <a href="#">ID3D11DeviceContext</a> .
ID3D10x	ID3D11x Many interfaces are identical between the two APIs.

## Other Related Technologies

10/10.1 Solution	11 Solution
HLSL Complier (D3D10Compile*, D3DX10Compile*) and shader reflection APIs	<p>D3DCompiler (see D3DCompiler.h)</p> <p>[!Note] For Windows Store apps, the <a href="#">D3DCompiler APIs</a> are supported only for development, not deployment.</p>

10/10.1 Solution	11 Solution
Effects 10	<p><a href="#">Effects 11</a> is available as shared source online.</p> <p>[!Note] This solution is not suited to Windows Store apps because it requires the <a href="#">D3DCompiler APIs</a> at runtime (deployment).</p>
D3DX9/D3DX10 Math	<a href="#">DirectXMath</a>
D3DX10	<p>D3DX11 in the legacy DirectX SDK <a href="#">DirectXTex</a>, <a href="#">DirectXTK</a>, and <a href="#">DirectXMesh</a> offer alternatives to many technologies in the legacy D3DX10 and D3DX11 libraries.</p> <p><a href="#">Direct2D</a> and <a href="#">DirectWrite</a> offer high-quality support for rendering styled lines and fonts.</p>

For info about the legacy DirectX SDK, see [Where is the DirectX SDK?](#).

## Direct3D 10.1 to Direct3D 11

Direct3D 10.1 is an extension of the Direct3D 10 interface, and all features of Direct3D 10.1 are available in Direct3D 11. Most of the porting from 10.1 to 11 is already addressed above moving from 10 to 11.

## Enumerations and Defines

Direct3D 10.1	Direct3D 11
D3D10_FEATURE_LEVEL1	<a href="#">D3D_FEATURE_LEVEL</a> Identical but defined in D3DCommon.h plus the addition of D3D_FEATURE_LEVEL_11_0 for 11 class hardware along with D3D_FEATURE_LEVEL_9_1, D3D_FEATURE_LEVEL_9_2, and D3D_FEATURE_LEVEL_9_3 for 10level9 (D3D10_FEATURE_LEVEL_9_1, D3D10_FEATURE_LEVEL_9_2, and D3D10_FEATURE_LEVEL_9_3 were also added for 10.1 to d3d10_1.h)

## Structures

Direct3D 10.1	Direct3D 11
D3D10_BLEND_DESC1	<a href="#">D3D11_BLEND_DESC</a> The 11 version is identical to 10.1.

Direct3D 10.1	Direct3D 11
D3D10_SHADER_RESOURCE_VIEW_DESC1	<a href="#">D3D11_SHADER_RESOURCE_VIEW_DESC</a> The 11 version is identical to 10.1.

## Interfaces

Direct3D 10.1	Direct3D 11
ID3D10Device1	<a href="#">ID3D11Device</a> and <a href="#">ID3D11DeviceContext</a> The device interface was split into these two parts. For quick porting you can make use of <a href="#">ID3D11Device::GetImmediateContext</a> . "ID3D10Device::GetTextFilterSize" and "SetTextFilerSize" methods no longer exist. See DirectWrite. Create*Shader takes an additional optional parameter for <a href="#">ID3D11ClassLinkage</a> . *SetShader and *GetShader take additional optional parameters for <a href="#">ID3D11ClassInstance</a> (s). <a href="#">CreateGeometryShaderWithStreamOutput</a> takes an array and count for multiple output stream strides. The limit for the NumEntries parameter of <a href="#">CreateGeometryShaderWithStreamOutput</a> has increased to D3D11_SO_STREAM_COUNT * D3D11_SO_OUTPUT_COMPONENT_COUNT in Direct3D 11.
ID3D10BlendState1	<a href="#">ID3D11BlendState</a>
ID3D10ShaderResourceView1	<a href="#">ID3D11ShaderResourceView</a>

## New Features for Direct3D 11

Once your code is updated to use the Direct3D 11 API, there are numerous [new features](#) to consider.

- Multithreaded rendering through command lists and multiple contexts
- Implementing advanced algorithms using Compute Shader (using 4.0, 4.1, or 5.0 shader profiles)
- New 11 class hardware features:
  - HLSL Shader Model 5.0
  - Dynamic Shader Linkage
  - Tessellation through Hull and Domain shaders
  - New block compression formats: BC6H for HDR images, BC7 for higher-fidelity standard images

- Utilizing [10level9 technology](#) for rendering on many Shader Model 2.0 and Shader Model 3.0 devices through the Direct3D 11 API for lower-end video hardware support on Windows Vista and Windows 7.
- Leveraging the WARP software rendering device.

## New Features for DirectX 11.1

Windows 8 includes further DirectX graphics enhancements to consider when you implement your DirectX graphics code, which include [Direct3D 11.1](#), [DXGI 1.2](#), [Windows Display Driver Model \(WDDM\) 1.2](#), feature level 11.1 hardware, Direct2D device contexts, and other improvements.

Partial support for [Direct3D 11.1](#) is available on Windows 7 as well via the [Platform Update for Windows 7](#), which is available through the [Platform Update for Windows 7](#).

## New Features for DirectX 11.2

The Windows 8.1 includes [Direct3D 11.2](#), [DXGI 1.3](#), and other improvements.

## Related topics

[Programming Guide for Direct3D 11](#)

[Effects \(Direct3D 11\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D Video Interfaces

Article • 08/19/2020

This topic lists the Direct3D video interfaces that are available in Direct3D 9Ex and that are supported on Windows 7 and later versions of Windows client operating systems and on Windows Server 2008 R2 and later versions of Windows server operating systems. You can use these interfaces and their methods to obtain the video content protection capabilities of the graphics driver and the overlay hardware capabilities of a Direct3D device. You can also use these interfaces and their methods to protect video content. These interfaces and their methods are defined in D3d9.h and described in the [Microsoft Media Foundation](#) section.

Interface	Description
<a href="#">IDirect3D9ExOverlayExtension</a>	Queries the overlay hardware capabilities of a Direct3D device.
<a href="#">IDirect3DAuthenticatedChannel9</a>	Provides a communication channel with the graphics driver or the Direct3D runtime.
<a href="#">IDirect3DCryptoSession9</a>	Represents a cryptographic session that is used to access a protected surface.
<a href="#">IDirect3DDevice9Video</a>	Enables an application to use content protection and encryption services that are implemented by a graphics driver.

## Related topics

[Effects \(Direct3D 11\)](#)

[Programming Guide for Direct3D 11](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11, utilities, and effects reference

Article • 02/20/2020

This section contains the reference pages for Direct3D 11-based graphics programming.

## In this section

Topic	Description
<a href="#">Direct3D 11 Reference</a>	The Direct3D 11 API is described in this section.
<a href="#">D3DCSX 11 Reference</a>	This section contains reference information about the D3DCSX utility library, which you can use with a compute shader.
<a href="#">D3DX 11 Reference</a>	The D3DX 11 API is described in this section.
<a href="#">Effects 11 Reference</a>	Use Effects 11 source to build your effects-type application. The Effects 11 source is available at <a href="#">Effects for Direct3D 11 Update</a> . The Effects 11 API is described in this section.

## Related topics

[Direct3D 11 Graphics](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11 Reference

Article • 08/23/2019

The Direct3D 11 API is described in this section.

## In this section

Topic	Description
Core Reference	The Direct3D API defines several core API elements.
Layer Reference	The Direct3D API defines several layer API elements.
Resource Reference	The Direct3D API defines several API elements to help you create and manage resources.
Shader Reference	The Direct3D API defines several API elements to help you create and manage programmable shaders. Shaders are executable programs that are programmed exclusively using HLSL.
10Level9 Reference	This section specifies the differences between each 10Level9 feature level and the D3D_FEATURE_LEVEL_11_0 and higher feature level for the <a href="#">ID3D11Device</a> and <a href="#">ID3D11DeviceContext</a> methods.
Direct3D 11 Return Codes	The return codes from API functions.
Common Version Reference	The Direct3D API defines several API elements that are common to the Direct3D 12, Direct3D 11, Direct3D 10, and Direct3D 10.1. You can use these API elements in your code for any of these Direct3D versions. These API elements are known as version neutral.
CD3D11 Helper Structures	Direct3D 11 defines several helper structures that you can use to create Direct3D structures. These helper structures behave like C++ classes.

## Related topics

[Direct3D 11 Reference](#)

[Direct3D 11 Graphics](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11 core reference

Article • 06/19/2021

The Direct3D API defines several core API elements.

## In this section

Topic	Description
Core Interfaces	This section contains information about the core interfaces.
Core Functions	This section contains information about the core functions.
Core Structures	This section contains information about the core structures.
Core Enumerations	This section contains information about the core enumerations.

## Related topics

[Direct3D 11 Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11 core interfaces

Article • 08/19/2021

This section contains information about the core interfaces.

## In this section

Topic	Description
<a href="#">ID3D11Asynchronous</a>	This interface encapsulates methods for retrieving data from the GPU asynchronously.
<a href="#">ID3D11BlendState</a>	The blend-state interface holds a description for blending state that you can bind to the <a href="#">output-merger stage</a> .
<a href="#">ID3D11BlendState1</a>	The blend-state interface holds a description for blending state that you can bind to the <a href="#">output-merger stage</a> . This blend-state interface supports logical operations as well as blending operations.
<a href="#">ID3D11CommandList</a>	The <a href="#">ID3D11CommandList</a> interface encapsulates a list of graphics commands for play back.
<a href="#">ID3D11Counter</a>	This interface encapsulates methods for measuring GPU performance.
<a href="#">ID3D11DepthStencilState</a>	The depth-stencil-state interface holds a description for depth-stencil state that you can bind to the <a href="#">output-merger stage</a> .
<a href="#">ID3D11Device</a>	The device interface represents a virtual adapter; it is used to create resources.
<a href="#">ID3D11Device1</a>	The device interface represents a virtual adapter; it is used to create resources. <a href="#">ID3D11Device1</a> adds new methods to those in <a href="#">ID3D11Device</a> .
<a href="#">ID3D11Device2</a>	The device interface represents a virtual adapter; it is used to create resources. <a href="#">ID3D11Device2</a> adds new methods to those in <a href="#">ID3D11Device1</a> .
<a href="#">ID3D11Device3</a>	The device interface represents a virtual adapter; it is used to create resources. <a href="#">ID3D11Device3</a> adds new methods to those in <a href="#">ID3D11Device2</a> .
<a href="#">ID3D11Device4</a>	The device interface represents a virtual adapter; it is used to create resources. <a href="#">ID3D11Device4</a> adds new methods to those in <a href="#">ID3D11Device3</a> , such as <a href="#">RegisterDeviceRemovedEvent</a> and <a href="#">UnregisterDeviceRemoved</a> .

Topic	Description
<a href="#">ID3D11Device5</a>	The device interface represents a virtual adapter; it is used to create resources. <a href="#">ID3D11Device5</a> adds new methods to those in <a href="#">ID3D11Device4</a> .
<a href="#">ID3D11DeviceChild</a>	A device-child interface accesses data used by a device.
<a href="#">ID3D11DeviceContext</a>	<p>The <a href="#">ID3D11DeviceContext</a> interface represents a device context which generates rendering commands.</p> <p>[!Note] The latest version of this interface is <a href="#">ID3D11DeviceContext4</a> introduced in the Windows 10 Creators Update. Applications targetting Windows 10 Creators Update should use the <a href="#">ID3D11DeviceContext4</a> interface instead of <a href="#">ID3D11Device</a>.</p>
<a href="#">ID3D11DeviceContext1</a>	The device context interface represents a device context; it is used to render commands. <a href="#">ID3D11DeviceContext1</a> adds new methods to those in <a href="#">ID3D11DeviceContext</a> .
<a href="#">ID3D11DeviceContext2</a>	The device context interface represents a device context; it is used to render commands. <a href="#">ID3D11DeviceContext2</a> adds new methods to those in <a href="#">ID3D11DeviceContext1</a> .
<a href="#">ID3D11DeviceContext3</a>	The device context interface represents a device context; it is used to render commands. <a href="#">ID3D11DeviceContext3</a> adds new methods to those in <a href="#">ID3D11DeviceContext2</a> .
<a href="#">ID3D11DeviceContext4</a>	The device context interface represents a device context; it is used to render commands. <a href="#">ID3D11DeviceContext4</a> adds new methods to those in <a href="#">ID3D11DeviceContext3</a> .
<a href="#">ID3DDeviceContextState</a>	The <a href="#">ID3DDeviceContextState</a> interface represents a context state object, which holds state and behavior information about a Microsoft Direct3D device.
<a href="#">ID3D11Fence</a>	Represents a fence, an object used for synchronization of the CPU and one or more GPUs.
<a href="#">ID3D11InputLayout</a>	An input-layout interface holds a definition of how to feed vertex data that is laid out in memory into the <a href="#">input-assembler stage</a> of the <a href="#">graphics pipeline</a> .
<a href="#">ID3D11Multithread</a>	Provides threading protection for critical sections of a multi-threaded application.
<a href="#">ID3D11Predicate</a>	A predicate interface determines whether geometry should be processed depending on the results of a previous draw call.

Topic	Description
<a href="#">ID3D11Query</a>	A query interface queries information from the GPU.
<a href="#">ID3D11Query1</a>	Represents a query object for querying information from the graphics processing unit (GPU).
<a href="#">ID3D11RasterizerState</a>	The rasterizer-state interface holds a description for rasterizer state that you can bind to the <a href="#">rasterizer stage</a> .
<a href="#">ID3D11RasterizerState1</a>	The rasterizer-state interface holds a description for rasterizer state that you can bind to the <a href="#">rasterizer stage</a> . This rasterizer-state interface supports forced sample count.
<a href="#">ID3D11RasterizerState2</a>	The rasterizer-state interface holds a description for rasterizer state that you can bind to the <a href="#">rasterizer stage</a> . This rasterizer-state interface supports forced sample count and conservative rasterization mode.
<a href="#">ID3D11SamplerState</a>	The sampler-state interface holds a description for sampler state that you can bind to any shader stage of the <a href="#">pipeline</a> for reference by texture sample operations.

Direct3D 11 implements interfaces for:

- [Resources](#)
- [Shaders](#)

## Related topics

[Core Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Asynchronous interface (d3d11.h)

Article 02/22/2024

This interface encapsulates methods for retrieving data from the GPU asynchronously.

## Inheritance

The **ID3D11Asynchronous** interface inherits from [ID3D11DeviceChild](#).

**ID3D11Asynchronous** also has these types of members:

## Methods

The **ID3D11Asynchronous** interface has these methods.

[+] [Expand table](#)

### [ID3D11Asynchronous::GetDataSize](#)

Get the size of the data (in bytes) that is output when calling [ID3D11DeviceContext::GetData](#).

## Remarks

There are three types of asynchronous interfaces, all of which inherit this interface:

- [ID3D11Query](#) - Queries information from the GPU.
- [ID3D11Predicate](#) - Determines whether a piece of geometry should be processed or not depending on the results of a previous draw call.
- [ID3D11Counter](#) - Measures GPU performance.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?



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

# ID3D11Asynchronous::GetDataSize method (d3d11.h)

Article 02/22/2024

Get the size of the data (in bytes) that is output when calling [ID3D11DeviceContext::GetData](#).

## Syntax

C++

```
UINT GetDataSize();
```

## Return value

Type: [UINT](#)

Size of the data (in bytes) that is output when calling `GetData`.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Asynchronous](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11BlendState interface (d3d11.h)

Article02/22/2024

The blend-state interface holds a description for blending state that you can bind to the [output-merger stage](#).

## Inheritance

The **ID3D11BlendState** interface inherits from [ID3D11DeviceChild](#). **ID3D11BlendState** also has these types of members:

## Methods

The **ID3D11BlendState** interface has these methods.

[+] [Expand table](#)

### [ID3D11BlendState::GetDesc](#)

Gets the description for blending state that you used to create the blend-state object.  
([ID3D11BlendState::GetDesc](#))

## Remarks

Blending applies a simple function to combine output values from a pixel shader with data in a render target. You have control over how the pixels are blended by using a predefined set of blending operations and preblending operations.

To create a blend-state object, call [ID3D11Device::CreateBlendState](#). To bind the blend-state object to the [output-merger stage](#), call [ID3D11DeviceContext::OMSetBlendState](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11BlendState::GetDesc method (d3d11.h)

Article 02/22/2024

Gets the description for blending state that you used to create the blend-state object.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_BLEND_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_BLEND\\_DESC\\*](#)

A pointer to a [D3D11\\_BLEND\\_DESC](#) structure that receives a description of the blend state.

## Return value

None

## Remarks

You use the description for blending state in a call to the [ID3D11Device::CreateBlendState](#) method to create the blend-state object.

## Requirements

[ Expand table]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11BlendState](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11BlendState1 interface (d3d11\_1.h)

Article 02/22/2024

The blend-state interface holds a description for blending state that you can bind to the [output-merger stage](#). This blend-state interface supports logical operations as well as blending operations.

## Inheritance

The **ID3D11BlendState1** interface inherits from [ID3D11BlendState](#). **ID3D11BlendState1** also has these types of members:

## Methods

The **ID3D11BlendState1** interface has these methods.

[+] [Expand table](#)

### [ID3D11BlendState1::GetDesc1](#)

Gets the description for blending state that you used to create the blend-state object.  
(`ID3D11BlendState1.GetDesc1`)

## Remarks

Blending applies a simple function to combine output values from a pixel shader with data in a render target. You have control over how the pixels are blended by using a predefined set of blending operations and preblending operations.

To create a blend-state object, call [ID3D11Device1::CreateBlendState1](#). To bind the blend-state object to the [output-merger stage](#), call [ID3D11DeviceContext::OMSetBlendState](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h

## See also

[Core Interfaces](#)

[ID3D11BlendState](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11BlendState1::GetDesc1 method (d3d11\_1.h)

Article02/22/2024

Gets the description for blending state that you used to create the blend-state object.

## Syntax

C++

```
void GetDesc1(  
    [out] D3D11_BLEND_DESC1 *pDesc  
);
```

## Parameters

[out] pDesc

A pointer to a [D3D11\\_BLEND\\_DESC1](#) structure that receives a description of the blend state. This blend state can specify logical operations as well as blending operations.

## Return value

None

## Remarks

You use the description for blending state in a call to the [ID3D11Device1::CreateBlendState1](#) method to create the blend-state object.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11BlendState1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11CommandList interface (d3d11.h)

Article 02/22/2024

The **ID3D11CommandList** interface encapsulates a list of graphics commands for playback.

## Inheritance

The **ID3D11CommandList** interface inherits from [ID3D11DeviceChild](#).

**ID3D11CommandList** also has these types of members:

## Methods

The **ID3D11CommandList** interface has these methods.

[+] Expand table

### [ID3D11CommandList::GetContextFlags](#)

Gets the initialization flags associated with the deferred context that created the command list.

## Remarks

There is no explicit creation method, simply declare an **ID3D11CommandList** interface, then call [ID3D11DeviceContext::FinishCommandList](#) to record commands or [ID3D11DeviceContext::ExecuteCommandList](#) to play back commands.

## Requirements

[+] Expand table

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

Requirement	Value
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11CommandList::GetContextFlags method (d3d11.h)

Article 02/22/2024

Gets the initialization flags associated with the deferred context that created the command list.

## Syntax

C++

```
UINT GetContextFlags();
```

## Return value

Type: [UINT](#)

The context flag is reserved for future use and is always 0.

## Remarks

The GetContextFlags method gets the flags that were supplied to the *ContextFlags* parameter of [ID3D11Device::CreateDeferredContext](#); however, the context flag is reserved for future use.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Counter interface (d3d11.h)

Article02/22/2024

This interface encapsulates methods for measuring GPU performance.

## Inheritance

The **ID3D11Counter** interface inherits from [ID3D11Asynchronous](#). **ID3D11Counter** also has these types of members:

## Methods

The **ID3D11Counter** interface has these methods.

[+] [Expand table](#)

<p><a href="#">ID3D11Counter::GetDesc</a></p> <p>Get a counter description. (<code>ID3D11Counter.GetDesc</code>)</p>

## Remarks

A counter can be created with [ID3D11Device::CreateCounter](#).

This is a derived class of [ID3D11Asynchronous](#).

Counter data is gathered by issuing an [ID3D11DeviceContext::Begin](#) command, issuing some graphics commands, issuing an [ID3D11DeviceContext::End](#) command, and then calling [ID3D11DeviceContext::GetData](#) to get data about what happened in between the Begin and End calls. The data returned by GetData will be different depending on the type of counter. The call to End causes the data returned by GetData to be accurate up until the last call to End.

Counters are best suited for profiling.

For a list of the types of performance counters, see [D3D11\\_COUNTER](#).

## Requirements

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11Asynchronous](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Counter::GetDesc method (d3d11.h)

Article 02/22/2024

Get a counter description.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_COUNTER_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_COUNTER\\_DESC\\*](#)

Pointer to a counter description (see [D3D11\\_COUNTER\\_DESC](#)).

## Return value

None

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DepthStencilState interface (d3d11.h)

Article 02/22/2024

The depth-stencil-state interface holds a description for depth-stencil state that you can bind to the [output-merger stage](#).

## Inheritance

The **ID3D11DepthStencilState** interface inherits from [ID3D11DeviceChild](#).

**ID3D11DepthStencilState** also has these types of members:

## Methods

The **ID3D11DepthStencilState** interface has these methods.

[+] [Expand table](#)

### [ID3D11DepthStencilState::GetDesc](#)

Gets the description for depth-stencil state that you used to create the depth-stencil-state object.

## Remarks

To create a depth-stencil-state object, call [ID3D11Device::CreateDepthStencilState](#). To bind the depth-stencil-state object to the [output-merger stage](#), call [ID3D11DeviceContext::OMSetDepthStencilState](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?



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

# ID3D11DepthStencilState::GetDesc method (d3d11.h)

Article 02/22/2024

Gets the description for depth-stencil state that you used to create the depth-stencil-state object.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_DEPTH_STENCIL_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_DEPTH\\_STENCIL\\_DESC\\*](#)

A pointer to a [D3D11\\_DEPTH\\_STENCIL\\_DESC](#) structure that receives a description of the depth-stencil state.

## Return value

None

## Remarks

You use the description for depth-stencil state in a call to the [ID3D11Device::CreateDepthStencilState](#) method to create the depth-stencil-state object.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DepthStencilState](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device interface (d3d11.h)

Article07/27/2022

The device interface represents a virtual adapter; it is used to create resources.

**Note** The latest version of this interface is **ID3D11Device5** introduced in the Windows 10 Creators Update. Applications targetting Windows 10 Creators Update should use the **ID3D11Device5** interface instead of **ID3D11Device**.

## Inheritance

The **ID3D11Device** interface inherits from the [IUnknown](#) interface. **ID3D11Device** also has these types of members:

## Methods

The **ID3D11Device** interface has these methods.

### [ID3D11Device::CheckCounter](#)

Get the type, name, units of measure, and a description of an existing counter.  
(`ID3D11Device.CheckCounter`)

### [ID3D11Device::CheckCounterInfo](#)

Get a counter's information. (`ID3D11Device.CheckCounterInfo`)

### [ID3D11Device::CheckFeatureSupport](#)

Gets information about the features that are supported by the current graphics driver.  
(`ID3D11Device.CheckFeatureSupport`)

### [ID3D11Device::CheckFormatSupport](#)

Get the support of a given format on the installed video device.  
(`ID3D11Device.CheckFormatSupport`)

[ID3D11Device::CheckMultisampleQualityLevels](#)

Get the number of quality levels available during multisampling.  
(ID3D11Device.CheckMultisampleQualityLevels)

[ID3D11Device::CreateBlendState](#)

Create a blend-state object that encapsulates blend state for the output-merger stage.  
(ID3D11Device.CreateBlendState)

[ID3D11Device::CreateBuffer](#)

Creates a buffer (vertex buffer, index buffer, or shader-constant buffer).

[ID3D11Device::CreateClassLinkage](#)

Creates class linkage libraries to enable dynamic shader linkage.

[ID3D11Device::CreateComputeShader](#)

Create a compute shader.

[ID3D11Device::CreateCounter](#)

Create a counter object for measuring GPU performance. (ID3D11Device.CreateCounter)

[ID3D11Device::CreateDeferredContext](#)

Creates a deferred context, which can record command lists.  
(ID3D11Device.CreateDeferredContext)

[ID3D11Device::CreateDepthStencilState](#)

Create a depth-stencil state object that encapsulates depth-stencil test information for the output-merger stage. (ID3D11Device.CreateDepthStencilState)

[ID3D11Device::CreateDepthStencilView](#)

Create a depth-stencil view for accessing resource data. (ID3D11Device.CreateDepthStencilView)

[ID3D11Device::CreateDomainShader](#)

Create a domain shader.

[ID3D11Device::CreateGeometryShader](#)

Create a geometry shader. (ID3D11Device.CreateGeometryShader)

[ID3D11Device::CreateGeometryShaderWithStreamOutput](#)

Creates a geometry shader that can write to streaming output buffers.  
(ID3D11Device.CreateGeometryShaderWithStreamOutput)

[ID3D11Device::CreateHullShader](#)

Create a hull shader.

[ID3D11Device::CreateInputLayout](#)

Create an input-layout object to describe the input-buffer data for the input-assembler stage.  
(ID3D11Device.CreateInputLayout)

[ID3D11Device::CreatePixelShader](#)

Create a pixel shader. (ID3D11Device.CreatePixelShader)

[ID3D11Device::CreatePredicate](#)

Creates a predicate. (ID3D11Device.CreatePredicate)

[ID3D11Device::CreateQuery](#)

This interface encapsulates methods for querying information from the GPU.  
(ID3D11Device.CreateQuery)

[ID3D11Device::CreateRasterizerState](#)

Create a rasterizer state object that tells the rasterizer stage how to behave.  
(ID3D11Device.CreateRasterizerState)

[ID3D11Device::CreateRenderTargetView](#)

Creates a render-target view for accessing resource data. (ID3D11Device.CreateRenderTargetView)

[ID3D11Device::CreateSamplerState](#)

Create a sampler-state object that encapsulates sampling information for a texture.  
(ID3D11Device.CreateSamplerState)

[ID3D11Device::CreateShaderResourceView](#)

Create a shader-resource view for accessing data in a resource.  
(ID3D11Device.CreateShaderResourceView)

[ID3D11Device::CreateTexture1D](#)

Creates an array of 1D textures.

[ID3D11Device::CreateTexture2D](#)

Create an array of 2D textures.

[ID3D11Device::CreateTexture3D](#)

Create a single 3D texture.

[ID3D11Device::CreateUnorderedAccessView](#)

Creates a view for accessing an unordered access resource.  
(`ID3D11Device.CreateUnorderedAccessView`)

[ID3D11Device::CreateVertexShader](#)

Create a vertex-shader object from a compiled shader. (`ID3D11Device.CreateVertexShader`)

[ID3D11Device::GetCreationFlags](#)

Get the flags used during the call to create the device with `D3D11CreateDevice`.

[ID3D11Device::GetDeviceRemovedReason](#)

Get the reason why the device was removed. (`ID3D11Device.GetDeviceRemovedReason`)

[ID3D11Device::GetExceptionMode](#)

Get the exception-mode flags. (`ID3D11Device.GetExceptionMode`)

[ID3D11Device::GetFeatureLevel](#)

Gets the feature level of the hardware device. (`ID3D11Device.GetFeatureLevel`)

[ID3D11Device::GetImmediateContext](#)

Gets an immediate context, which can play back command lists.  
(`ID3D11Device.GetImmediateContext`)

[ID3D11Device::GetPrivateData](#)

Get application-defined data from a device.

[ID3D11Device::OpenSharedResource](#)

Give a device access to a shared resource created on a different device.

[ID3D11Device::SetExceptionMode](#)

Get the exception-mode flags. (`ID3D11Device.SetExceptionMode`)

## [ID3D11Device::SetPrivateData](#)

Set data to a device and associate that data with a guid. (`ID3D11Device.SetPrivateData`)

## [ID3D11Device::SetPrivateDataInterface](#)

Associate an `IUnknown`-derived interface with this device child and associate that interface with an application-defined guid. (`ID3D11Device.SetPrivateDataInterface`)

## Remarks

A device is created using [D3D11CreateDevice](#).

**Windows Phone 8:** This API is supported.

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[IUnknown](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CheckCounter method (d3d11.h)

Article 07/27/2022

Get the type, name, units of measure, and a description of an existing counter.

## Syntax

C++

```
HRESULT CheckCounter(
    [in]                  const D3D11_COUNTER_DESC *pDesc,
    [out]                 D3D11_COUNTER_TYPE      *pType,
    [out]                 UINT                *pActiveCounters,
    [out, optional]       LPSTR               szName,
    [in, out, optional]  UINT                *pNameLength,
    [out, optional]       LPSTR               szUnits,
    [in, out, optional]  UINT                *pUnitsLength,
    [out, optional]       LPSTR               szDescription,
    [in, out, optional]  UINT                *pDescriptionLength
);
```

## Parameters

[in] pDesc

Type: [const D3D11\\_COUNTER\\_DESC\\*](#)

Pointer to a counter description (see [D3D11\\_COUNTER\\_DESC](#)). Specifies which counter information is to be retrieved about.

[out] pType

Type: [D3D11\\_COUNTER\\_TYPE\\*](#)

Pointer to the data type of a counter (see [D3D11\\_COUNTER\\_TYPE](#)). Specifies the data type of the counter being retrieved.

[out] pActiveCounters

Type: [UINT\\*](#)

Pointer to the number of hardware counters that are needed for this counter type to be created. All instances of the same counter type use the same hardware counters.

[out, optional] szName

Type: **LPSTR**

String to be filled with a brief name for the counter. May be **NULL** if the application is not interested in the name of the counter.

[in, out, optional] pNameLength

Type: **UINT\***

Length of the string returned to szName. Can be **NULL**.

[out, optional] szUnits

Type: **LPSTR**

Name of the units a counter measures, provided the memory the pointer points to has enough room to hold the string. Can be **NULL**. The returned string will always be in English.

[in, out, optional] pUnitsLength

Type: **UINT\***

Length of the string returned to szUnits. Can be **NULL**.

[out, optional] szDescription

Type: **LPSTR**

A description of the counter, provided the memory the pointer points to has enough room to hold the string. Can be **NULL**. The returned string will always be in English.

[in, out, optional] pDescriptionLength

Type: **UINT\***

Length of the string returned to szDescription. Can be **NULL**.

## Return value

Type: **HRESULT**

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Length parameters can be **NULL**, which indicates the application is not interested in the length nor the corresponding string value. When a length parameter is non-**NULL** and the corresponding string is **NULL**, the input value of the length parameter is ignored, and the length of the corresponding string (including terminating **NULL**) will be returned through the length parameter. When length and the corresponding parameter are both non-**NULL**, the input value of length is checked to ensure there is enough room, and then the length of the string (including terminating **NULL** character) is passed out through the length parameter.

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CheckCounterInfo method (d3d11.h)

Article 02/22/2024

Get a counter's information.

## Syntax

C++

```
void CheckCounterInfo(  
    [out] D3D11_COUNTER_INFO *pCounterInfo  
);
```

## Parameters

[out] pCounterInfo

Type: [D3D11\\_COUNTER\\_INFO\\*](#)

Pointer to counter information (see [D3D11\\_COUNTER\\_INFO](#)).

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CheckFeatureSupport method (d3d11.h)

Article 07/27/2022

Gets information about the features that are supported by the current graphics driver.

## Syntax

C++

```
HRESULT CheckFeatureSupport(  
    D3D11_FEATURE Feature,  
    [out] void* pFeatureSupportData,  
    UINT FeatureSupportDataSize  
>;
```

## Parameters

Feature

Type: [D3D11\\_FEATURE](#)

A member of the [D3D11\\_FEATURE](#) enumerated type that describes which feature to query for support.

[out] pFeatureSupportData

Type: [void\\*](#)

Upon completion of the method, the passed structure is filled with data that describes the feature support.

FeatureSupportDataSize

Type: [UINT](#)

The size of the structure passed to the *pFeatureSupportData* parameter.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns E\_INVALIDARG if an unsupported data type is passed to the *pFeatureSupportData* parameter or a size mismatch is detected for the *FeatureSupportDataSize* parameter.

## Remarks

To query for multi-threading support, pass the D3D11\_FEATURE\_THREADING value to the *Feature* parameter, pass the D3D11\_FEATURE\_DATA\_THREADING structure to the *pFeatureSupportData* parameter, and pass the size of the D3D11\_FEATURE\_DATA\_THREADING structure to the *FeatureSupportDataSize* parameter.

Calling CheckFeatureSupport with *Feature* set to D3D11\_FEATURE\_FORMAT\_SUPPORT causes the method to return the same information that would be returned by [ID3D11Device::CheckFormatSupport](#).

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CheckFormatSupport method (d3d11.h)

Article 02/22/2024

Get the support of a given format on the installed video device.

## Syntax

C++

```
HRESULT CheckFormatSupport(
    [in] DXGI_FORMAT Format,
    [out] UINT       *pFormatSupport
);
```

## Parameters

[in] Format

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#) enumeration that describes a format for which to check for support.

[out] pFormatSupport

Type: [UINT\\*](#)

A bitfield of [D3D11\\_FORMAT\\_SUPPORT](#) enumeration values describing how the specified format is supported on the installed device. The values are ORed together.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns E\_INVALIDARG if the *Format* parameter is NULL, or returns E\_FAIL if the described format does not exist.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CheckMultisampleQualityLevels method (d3d11.h)

Article 02/22/2024

Get the number of quality levels available during multisampling.

## Syntax

C++

```
HRESULT CheckMultisampleQualityLevels(
    [in] DXGI_FORMAT Format,
    [in] UINT SampleCount,
    [out] UINT *pNumQualityLevels
);
```

## Parameters

[in] Format

Type: [DXGI\\_FORMAT](#)

The texture format. See [DXGI\\_FORMAT](#).

[in] SampleCount

Type: [UINT](#)

The number of samples during multisampling.

[out] pNumQualityLevels

Type: [UINT\\*](#)

Number of quality levels supported by the adapter. See [Remarks](#).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

When multisampling a texture, the number of quality levels available for an adapter is dependent on the texture format used and the number of samples requested. The maximum number of quality levels is defined by `D3D11_MAX_MULTISAMPLE_SAMPLE_COUNT` in `d3d11.h`. If this method returns 0 (`S_OK`), and the output parameter `pNumQualityLevels` receives a positive value, then the format and sample count combination is supported for the device. When the combination is not supported, this method returns a failure `HRESULT` code (that is, a negative integer), or sets `pNumQualityLevels` output parameter to zero, or both.

Furthermore, the definition of a quality level is left to each hardware vendor to define; however no facility is provided by Direct3D to help discover this information.

Note that `FEATURE_LEVEL_10_1` devices are required to support 4x MSAA for all render targets except `R32G32B32A32` and `R32G32B32`. `FEATURE_LEVEL_11_0` devices are required to support 4x MSAA for all render target formats, and 8x MSAA for all render target formats except `R32G32B32A32` formats.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	<code>d3d11.h</code>
Library	<code>D3D11.lib</code>

## See also

[ID3D11Device](#)

## Feedback

Was this page helpful?

 Yes

 No

# ID3D11Device::CreateBlendState method (d3d11.h)

Article 02/22/2024

Create a blend-state object that encapsulates blend state for the output-merger stage.

## Syntax

C++

```
HRESULT CreateBlendState(
    [in]          const D3D11_BLEND_DESC *pBlendStateDesc,
    [out, optional] ID3D11BlendState     **ppBlendState
);
```

## Parameters

[in] pBlendStateDesc

Type: [const D3D11\\_BLEND\\_DESC\\*](#)

Pointer to a blend-state description (see [D3D11\\_BLEND\\_DESC](#)).

[out, optional] ppBlendState

Type: [ID3D11BlendState\\*\\*](#)

Address of a pointer to the blend-state object created (see [ID3D11BlendState](#)).

## Return value

Type: [HRESULT](#)

This method returns [E\\_OUTOFMEMORY](#) if there is insufficient memory to create the blend-state object. See [Direct3D 11 Return Codes](#) for other possible return values.

## Remarks

An application can create up to 4096 unique blend-state objects. For each object created, the runtime checks to see if a previous object has the same state. If such a

previous object exists, the runtime will return a pointer to previous instance instead of creating a duplicate object.

**Windows Phone 8:** This API is supported.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateBuffer method (d3d11.h)

Article 10/13/2021

Creates a buffer (vertex buffer, index buffer, or shader-constant buffer).

## Syntax

C++

```
HRESULT CreateBuffer(
    [in]           const D3D11_BUFFER_DESC      *pDesc,
    [in, optional] const D3D11_SUBRESOURCE_DATA *pInitialData,
    [out, optional] ID3D11Buffer              **ppBuffer
);
```

## Parameters

[in] pDesc

Type: **const D3D11\_BUFFER\_DESC\***

A pointer to a **D3D11\_BUFFER\_DESC** structure that describes the buffer.

[in, optional] pInitialData

Type: **const D3D11\_SUBRESOURCE\_DATA\***

A pointer to a **D3D11\_SUBRESOURCE\_DATA** structure that describes the initialization data; use **NULL** to allocate space only (with the exception that it cannot be **NULL** if the usage flag is **D3D11\_USAGE\_IMMUTABLE**).

If you don't pass anything to *pInitialData*, the initial content of the memory for the buffer is undefined. In this case, you need to write the buffer content some other way before the resource is read.

[out, optional] ppBuffer

Type: **ID3D11Buffer\*\***

Address of a pointer to the **ID3D11Buffer** interface for the buffer object created. Set this parameter to **NULL** to validate the other input parameters (**S\_FALSE** indicates a pass).

# Return value

Type: [HRESULT](#)

This method returns [E\\_OUTOFMEMORY](#) if there is insufficient memory to create the buffer. See [Direct3D 11 Return Codes](#) for other possible return values.

## Remarks

For example code, see [How to: Create a Vertex Buffer](#), [How to: Create an Index Buffer](#) or [How to: Create a Constant Buffer](#).

For a constant buffer ([BindFlags](#) of [D3D11\\_BUFFER\\_DESC](#) set to [D3D11\\_BIND\\_CONSTANT\\_BUFFER](#)), you must set the [ByteWidth](#) value of [D3D11\\_BUFFER\\_DESC](#) in multiples of 16, and less than or equal to [D3D11\\_REQ\\_CONSTANT\\_BUFFER\\_ELEMENT\\_COUNT](#).

The Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems, provides the following new functionality for [CreateBuffer](#):

You can create a constant buffer that is larger than the maximum constant buffer size that a shader can access (4096 32-bit\*4-component constants – 64KB). When you bind the constant buffer to the pipeline (for example, via [PSSetConstantBuffers](#) or [PSSetConstantBuffers1](#)), you can define a range of the buffer that the shader can access that fits within the 4096 constant limit.

The Direct3D 11.1 runtime (available in Windows 8 and later operating systems) emulates this feature for [feature level](#) 9.1, 9.2, and 9.3; therefore, this feature is supported for feature level 9.1, 9.2, and 9.3.

This feature is always available on new drivers for feature level 10 and higher.

On runtimes older than Direct3D 11.1, a call to [CreateBuffer](#) to request a constant buffer that is larger than 4096 fails.

## Requirements

Target Platform	Windows
Header	d3d11.h

Library

D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateClassLinkage method (d3d11.h)

Article 02/22/2024

Creates class linkage libraries to enable dynamic shader linkage.

## Syntax

C++

```
HRESULT CreateClassLinkage(
    [out] ID3D11ClassLinkage **ppLinkage
);
```

## Parameters

[out] ppLinkage

Type: [ID3D11ClassLinkage\\*\\*](#)

A pointer to a class-linkage interface pointer (see [ID3D11ClassLinkage](#)).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

The [ID3D11ClassLinkage](#) interface returned in *ppLinkage* is associated with a shader by passing it as a parameter to one of the [ID3D11Device](#) create shader methods such as [ID3D11Device::CreatePixelShader](#).

## Examples

Using CreateClassLinkage

```
ID3D11ClassLinkage * g_pPSClassLinkage = NULL;  
pd3dDevice->CreateClassLinkage( &g_pPSClassLinkage );
```

# Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateComputeShader method (d3d11.h)

Article 02/22/2024

Create a [compute shader](#).

## Syntax

C++

```
HRESULT CreateComputeShader(
    [in]          const void      *pShaderBytecode,
    [in]          SIZE_T          BytecodeLength,
    [in, optional] ID3D11ClassLinkage *pClassLinkage,
    [out, optional] ID3D11ComputeShader **ppComputeShader
);
```

## Parameters

[in] `pShaderBytecode`

Type: `const void*`

A pointer to a compiled shader.

[in] `BytecodeLength`

Type: `SIZE_T`

Size of the compiled shader in `pShaderBytecode`.

[in, optional] `pClassLinkage`

Type: [`ID3D11ClassLinkage`\\*](#)

A pointer to a [ID3D11ClassLinkage](#), which represents class linkage interface; the value can be **NULL**.

[out, optional] `ppComputeShader`

Type: [`ID3D11ComputeShader`\\*\\*](#)

Address of a pointer to an [ID3D11ComputeShader](#) interface. If this is **NULL**, all other parameters will be validated; if validation passes, CreateComputeShader returns **S\_FALSE** instead of **S\_OK**.

## Return value

Type: [HRESULT](#)

This method returns **E\_OUTOFMEMORY** if there is insufficient memory to create the compute shader.

See [Direct3D 11 Return Codes](#) for other possible return values.

## Remarks

For an example, see [How To: Create a Compute Shader](#) and [HDRToneMappingCS11 Sample](#).

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateCounter method (d3d11.h)

Article 07/27/2022

Create a counter object for measuring GPU performance.

## Syntax

C++

```
HRESULT CreateCounter(
    [in]           const D3D11_COUNTER_DESC *pCounterDesc,
    [out, optional] ID3D11Counter        **ppCounter
);
```

## Parameters

[in] pCounterDesc

Type: [const D3D11\\_COUNTER\\_DESC\\*](#)

Pointer to a counter description (see [D3D11\\_COUNTER\\_DESC](#)).

[out, optional] ppCounter

Type: [ID3D11Counter\\*\\*](#)

Address of a pointer to a counter (see [ID3D11Counter](#)).

## Return value

Type: [HRESULT](#)

If this function succeeds, it will return S\_OK. If it fails, possible return values are: S\_FALSE, E\_OUTOFMEMORY, DXGI\_ERROR\_UNSUPPORTED, DXGI\_ERROR\_NONEXCLUSIVE, or E\_INVALIDARG.

DXGI\_ERROR\_UNSUPPORTED is returned whenever the application requests to create a well-known counter, but the current device does not support it.

`DXGI_ERROR_NONEXCLUSIVE` indicates that another device object is currently using the counters, so they cannot be used by this device at the moment.

`E_INVALIDARG` is returned whenever an out-of-range well-known or device-dependent counter is requested, or when the simultaneously active counters have been exhausted.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateDeferredContext method (d3d11.h)

Article 02/22/2024

Creates a deferred context, which can record command lists.

## Syntax

C++

```
HRESULT CreateDeferredContext(
    UINT             ContextFlags,
    [out, optional] ID3D11DeviceContext **ppDeferredContext
);
```

## Parameters

ContextFlags

Type: [UINT](#)

Reserved for future use. Pass 0.

[out, optional] ppDeferredContext

Type: [ID3D11DeviceContext\\*\\*](#)

Upon completion of the method, the passed pointer to an [ID3D11DeviceContext](#) interface pointer is initialized.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the following:

- Returns DXGI\_ERROR\_DEVICE\_REMOVED if the video card has been physically removed from the system, or a driver upgrade for the video card has occurred. If this error occurs, you should destroy and recreate the device.
- Returns DXGI\_ERROR\_INVALID\_CALL if the [CreateDeferredContext](#) method cannot be called from the current context. For example, if the device was created

with the [D3D11\\_CREATE\\_DEVICE\\_SINGLETHREADED](#) value, [CreateDeferredContext](#) returns [DXGI\\_ERROR\\_INVALID\\_CALL](#).

- Returns [E\\_INVALIDARG](#) if the *ContextFlags* parameter is invalid.
- Returns [E\\_OUTOFMEMORY](#) if the application has exhausted available memory.

## Remarks

A deferred context is a thread-safe context that you can use to record graphics commands on a thread other than the main rendering thread. Using a deferred context, you can record graphics commands into a command list that is encapsulated by the [ID3D11CommandList](#) interface. After all scene items are recorded, you can then submit them to the main render thread for final rendering. In this manner, you can perform rendering tasks concurrently across multiple threads and potentially improve performance in multi-core CPU scenarios.

You can create multiple deferred contexts.

**Note** If you use the [D3D11\\_CREATE\\_DEVICE\\_SINGLETHREADED](#) value to create the device that is represented by [ID3D11Device](#), the [CreateDeferredContext](#) method will fail, and you will not be able to create a deferred context.

For more information about deferred contexts, see [Immediate and Deferred Rendering](#).

**Windows Phone 8:** This API is supported.

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11Device](#)

[ID3D11Device1::CreateDeferredContext1](#)

[ID3D11Device2::CreateDeferredContext2](#)

[ID3D11Device3::CreateDeferredContext3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateDepthStencilState method (d3d11.h)

Article 02/22/2024

Create a depth-stencil state object that encapsulates depth-stencil test information for the output-merger stage.

## Syntax

C++

```
HRESULT CreateDepthStencilState(  
    [in]           const D3D11_DEPTH_STENCIL_DESC *pDepthStencilDesc,  
    [out, optional] ID3D11DepthStencilState        **ppDepthStencilState  
) ;
```

## Parameters

[in] pDepthStencilDesc

Type: [const D3D11\\_DEPTH\\_STENCIL\\_DESC\\*](#)

Pointer to a depth-stencil state description (see [D3D11\\_DEPTH\\_STENCIL\\_DESC](#)).

[out, optional] ppDepthStencilState

Type: [ID3D11DepthStencilState\\*\\*](#)

Address of a pointer to the depth-stencil state object created (see [ID3D11DepthStencilState](#)).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

4096 unique depth-stencil state objects can be created on a device at a time.

If an application attempts to create a depth-stencil-state interface with the same state as an existing interface, the same interface will be returned and the total number of unique depth-stencil state objects will stay the same.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateDepthStencilView method (d3d11.h)

Article 02/22/2024

Create a depth-stencil view for accessing resource data.

## Syntax

C++

```
HRESULT CreateDepthStencilView(
    [in]          ID3D11Resource           *pResource,
    [in, optional] const D3D11_DEPTH_STENCIL_VIEW_DESC *pDesc,
    [out, optional] ID3D11DepthStencilView        **ppDepthStencilView
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

Pointer to the resource that will serve as the depth-stencil surface. This resource must have been created with the [D3D11\\_BIND\\_DEPTH\\_STENCIL](#) flag.

[in, optional] pDesc

Type: [const D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC\\*](#)

Pointer to a depth-stencil-view description (see [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)). Set this parameter to **NULL** to create a view that accesses mipmap level 0 of the entire resource (using the format the resource was created with).

[out, optional] ppDepthStencilView

Type: [ID3D11DepthStencilView\\*\\*](#)

Address of a pointer to an [ID3D11DepthStencilView](#). Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

# Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

A depth-stencil view can be bound to the output-merger stage by calling [ID3D11DeviceContext::OMSetRenderTargets](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateDomainShader method (d3d11.h)

Article 02/22/2024

Create a [domain shader](#).

## Syntax

C++

```
HRESULT CreateDomainShader(
    [in]          const void      *pShaderBytecode,
    [in]          SIZE_T          BytecodeLength,
    [in, optional] ID3D11ClassLinkage *pClassLinkage,
    [out, optional] ID3D11DomainShader **ppDomainShader
);
```

## Parameters

[in] pShaderBytecode

Type: **const void\***

A pointer to a compiled shader.

[in] BytecodeLength

Type: **SIZE\_T**

Size of the compiled shader.

[in, optional] pClassLinkage

Type: [\*\*ID3D11ClassLinkage\\*\*\*](#)

A pointer to a class linkage interface (see [ID3D11ClassLinkage](#)); the value can be **NULL**.

[out, optional] ppDomainShader

Type: [\*\*ID3D11DomainShader\\*\\*\*\*](#)

Address of a pointer to a [ID3D11DomainShader](#) interface. If this is **NULL**, all other parameters will be validated, and if all parameters pass validation this API will return

`S_FALSE` instead of `S_OK`.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

The Direct3D 11.1 runtime, which is available starting with Windows 8, provides the following new functionality for [CreateDomainShader](#).

The following shader model 5.0 instructions are available to just pixel shaders and compute shaders in the Direct3D 11.0 runtime. For the Direct3D 11.1 runtime, because unordered access views (UAV) are available at all shader stages, you can use these instructions in all shader stages.

Therefore, if you use the following shader model 5.0 instructions in a domain shader, you can successfully pass the compiled domain shader to `pShaderBytecode`. That is, the call to [CreateDomainShader](#) succeeds.

If you pass a compiled shader to `pShaderBytecode` that uses any of the following instructions on a device that doesn't support UAVs at every shader stage (including existing drivers that are not implemented to support UAVs at every shader stage), [CreateDomainShader](#) fails. [CreateDomainShader](#) also fails if the shader tries to use a UAV slot beyond the set of UAV slots that the hardware supports.

- [dcl\\_uav\\_typed](#)
- [dcl\\_uav\\_raw](#)
- [dcl\\_uav\\_structured](#)
- [ld\\_raw](#)
- [ld\\_structured](#)
- [ld\\_uav\\_typed](#)
- [store\\_raw](#)
- [store\\_structured](#)
- [store\\_uav\\_typed](#)
- [sync\\_uglobal](#)
- All atomics and immediate atomics (for example, [atomic\\_and](#) and [imm\\_atomic\\_and](#))

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateGeometryShader method (d3d11.h)

Article 02/22/2024

Create a geometry shader.

## Syntax

C++

```
HRESULT CreateGeometryShader(
    [in]          const void      *pShaderBytecode,
    [in]          SIZE_T          BytecodeLength,
    [in, optional] ID3D11ClassLinkage *pClassLinkage,
    [out, optional] ID3D11GeometryShader **ppGeometryShader
);
```

## Parameters

[in] pShaderBytecode

Type: **const void\***

A pointer to the compiled shader.

[in] BytecodeLength

Type: **SIZE\_T**

Size of the compiled geometry shader.

[in, optional] pClassLinkage

Type: **ID3D11ClassLinkage\***

A pointer to a class linkage interface (see [ID3D11ClassLinkage](#)); the value can be **NULL**.

[out, optional] ppGeometryShader

Type: **ID3D11GeometryShader\*\***

Address of a pointer to a [ID3D11GeometryShader](#) interface. If this is **NULL**, all other parameters will be validated, and if all parameters pass validation this API will return

`S_FALSE` instead of `S_OK`.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

After it is created, the shader can be set to the device by calling [ID3D11DeviceContext::GSSetShader](#).

The Direct3D 11.1 runtime, which is available starting with Windows 8, provides the following new functionality for [CreateGeometryShader](#).

The following shader model 5.0 instructions are available to just pixel shaders and compute shaders in the Direct3D 11.0 runtime. For the Direct3D 11.1 runtime, because unordered access views (UAV) are available at all shader stages, you can use these instructions in all shader stages.

Therefore, if you use the following shader model 5.0 instructions in a geometry shader, you can successfully pass the compiled geometry shader to *pShaderBytecode*. That is, the call to [CreateGeometryShader](#) succeeds.

If you pass a compiled shader to *pShaderBytecode* that uses any of the following instructions on a device that doesn't support UAVs at every shader stage (including existing drivers that are not implemented to support UAVs at every shader stage), [CreateGeometryShader](#) fails. [CreateGeometryShader](#) also fails if the shader tries to use a UAV slot beyond the set of UAV slots that the hardware supports.

- [dcl\\_uav\\_typed](#)
- [dcl\\_uav\\_raw](#)
- [dcl\\_uav\\_structured](#)
- [ld\\_raw](#)
- [ld\\_structured](#)
- [ld\\_uav\\_typed](#)
- [store\\_raw](#)
- [store\\_structured](#)
- [store\\_uav\\_typed](#)
- [sync\\_uglobal](#)

- All atomics and immediate atomics (for example, [atomic\\_and](#) and [imm\\_atomic\\_and](#))

## Examples

### Usage Example

```
ID3D11GeometryShader* g_pGeometryShader11 = NULL;
ID3DBlob* pGeometryShaderBuffer = NULL;
ID3DBlob * errorbuffer = NULL;

D3DX11CompileFromFile( str, NULL, NULL, "GS", "gs_4_0", dwShaderFlags, 0,
NULL,
&pGeometryShaderBuffer,
&errorbuffer, NULL );

pd3dDevice->CreateGeometryShader( pGeometryShaderBuffer->GetBufferPointer(),
pGeometryShaderBuffer->GetBufferSize(), NULL,
&g_pGeometryShader11 );
```

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11Device::CreateGeometryShaderWithStreamOutput method (d3d11.h)

Article 07/27/2022

Creates a geometry shader that can write to streaming output buffers.

## Syntax

C++

```
HRESULT CreateGeometryShaderWithStreamOutput(
    [in]          const void*           *pShaderBytecode,
    [in]          SIZE_T              BytecodeLength,
    [in, optional] const D3D11_SO_DECLARATION_ENTRY* pSODDeclaration,
    [in]          UINT                NumEntries,
    [in, optional] const UINT*        pBufferStrides,
    [in]          UINT                NumStrides,
    [in]          UINT                RasterizedStream,
    [in, optional] ID3D11ClassLinkage* pClassLinkage,
    [out, optional] ID3D11GeometryShader** ppGeometryShader
);
```

## Parameters

[in] `pShaderBytecode`

Type: `const void*`

A pointer to the compiled geometry shader for a standard geometry shader plus stream output. For info on how to get this pointer, see [Getting a Pointer to a Compiled Shader](#).

To create the stream output without using a geometry shader, pass a pointer to the output signature for the prior stage. To obtain this output signature, call the [D3DGetOutputSignatureBlob](#) compiler function. You can also pass a pointer to the compiled shader for the prior stage (for example, the [vertex-shader stage](#) or [domain-shader stage](#)). This compiled shader provides the output signature for the data.

[in] `BytecodeLength`

Type: `SIZE_T`

Size of the compiled geometry shader.

[in, optional] pSODeclaration

Type: **const D3D11\_SO\_DECLARATION\_ENTRY\***

Pointer to a [D3D11\\_SO\\_DECLARATION\\_ENTRY](#) array. Cannot be **NULL** if NumEntries > 0.

[in] NumEntries

Type: **UINT**

The number of entries in the stream output declaration ( ranges from 0 to [D3D11\\_SO\\_STREAM\\_COUNT \\* D3D11\\_SO\\_OUTPUT\\_COMPONENT\\_COUNT](#) ).

[in, optional] pBufferStrides

Type: **const UINT\***

An array of buffer strides; each stride is the size of an element for that buffer.

[in] NumStrides

Type: **UINT**

The number of strides (or buffers) in *pBufferStrides* (ranges from 0 to [D3D11\\_SO\\_BUFFER\\_SLOT\\_COUNT](#)).

[in] RasterizedStream

Type: **UINT**

The index number of the stream to be sent to the rasterizer stage (ranges from 0 to [D3D11\\_SO\\_STREAM\\_COUNT - 1](#)). Set to [D3D11\\_SO\\_NO\\_RASTERIZED\\_STREAM](#) if no stream is to be rasterized.

[in, optional] pClassLinkage

Type: **ID3D11ClassLinkage\***

A pointer to a class linkage interface (see [ID3D11ClassLinkage](#)); the value can be **NULL**.

[out, optional] ppGeometryShader

Type: **ID3D11GeometryShader\*\***

Address of a pointer to an [ID3D11GeometryShader](#) interface, representing the geometry shader that was created. Set this to **NULL** to validate the other parameters; if validation passes, the method will return [S\\_FALSE](#) instead of [S\\_OK](#).

# Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

For more info about using `CreateGeometryShaderWithStreamOutput`, see [Create a Geometry-Shader Object with Stream Output](#).

The Direct3D 11.1 runtime, which is available starting with Windows 8, provides the following new functionality for `CreateGeometryShaderWithStreamOutput`.

The following shader model 5.0 instructions are available to just pixel shaders and compute shaders in the Direct3D 11.0 runtime. For the Direct3D 11.1 runtime, because unordered access views (UAV) are available at all shader stages, you can use these instructions in all shader stages.

Therefore, if you use the following shader model 5.0 instructions in a geometry shader, you can successfully pass the compiled geometry shader to *pShaderBytecode*. That is, the call to `CreateGeometryShaderWithStreamOutput` succeeds.

If you pass a compiled shader to *pShaderBytecode* that uses any of the following instructions on a device that doesn't support UAVs at every shader stage (including existing drivers that are not implemented to support UAVs at every shader stage), `CreateGeometryShaderWithStreamOutput` fails.

`CreateGeometryShaderWithStreamOutput` also fails if the shader tries to use a UAV slot beyond the set of UAV slots that the hardware supports.

- [dcl\\_uav\\_typed](#)
- [dcl\\_uav\\_raw](#)
- [dcl\\_uav\\_structured](#)
- [ld\\_raw](#)
- [ld\\_structured](#)
- [ld\\_uav\\_typed](#)
- [store\\_raw](#)
- [store\\_structured](#)
- [store\\_uav\\_typed](#)
- [sync\\_uglobal](#)
- All atomics and immediate atomics (for example, [atomic\\_and](#) and [imm\\_atomic\\_and](#))

Windows Phone 8: This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateHullShader method (d3d11.h)

Article 02/22/2024

Create a [hull shader](#).

## Syntax

C++

```
HRESULT CreateHullShader(
    [in]          const void      *pShaderBytecode,
    [in]          SIZE_T          BytecodeLength,
    [in, optional] ID3D11ClassLinkage *pClassLinkage,
    [out, optional] ID3D11HullShader   **ppHullShader
);
```

## Parameters

[in] pShaderBytecode

Type: **const void\***

A pointer to a compiled shader.

[in] BytecodeLength

Type: **SIZE\_T**

Size of the compiled shader.

[in, optional] pClassLinkage

Type: [\*\*ID3D11ClassLinkage\\*\*\*](#)

A pointer to a class linkage interface (see [ID3D11ClassLinkage](#)); the value can be **NULL**.

[out, optional] ppHullShader

Type: [\*\*ID3D11HullShader\\*\\*\*\*](#)

Address of a pointer to a [ID3D11HullShader](#) interface.

# Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

The Direct3D 11.1 runtime, which is available starting with Windows 8, provides the following new functionality for [CreateHullShader](#).

The following shader model 5.0 instructions are available to just pixel shaders and compute shaders in the Direct3D 11.0 runtime. For the Direct3D 11.1 runtime, because unordered access views (UAV) are available at all shader stages, you can use these instructions in all shader stages.

Therefore, if you use the following shader model 5.0 instructions in a hull shader, you can successfully pass the compiled hull shader to *pShaderBytecode*. That is, the call to [CreateHullShader](#) succeeds.

If you pass a compiled shader to *pShaderBytecode* that uses any of the following instructions on a device that doesn't support UAVs at every shader stage (including existing drivers that are not implemented to support UAVs at every shader stage), [CreateHullShader](#) fails. [CreateHullShader](#) also fails if the shader tries to use a UAV slot beyond the set of UAV slots that the hardware supports.

- [dcl\\_uav\\_typed](#)
- [dcl\\_uav\\_raw](#)
- [dcl\\_uav\\_structured](#)
- [ld\\_raw](#)
- [ld\\_structured](#)
- [ld\\_uav\\_typed](#)
- [store\\_raw](#)
- [store\\_structured](#)
- [store\\_uav\\_typed](#)
- [sync\\_uglobal](#)
- All atomics and immediate atomics (for example, [atomic\\_and](#) and [imm\\_atomic\\_and](#))

## Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateInputLayout method (d3d11.h)

Article 02/22/2024

Create an input-layout object to describe the input-buffer data for the input-assembler stage.

## Syntax

C++

```
HRESULT CreateInputLayout(
    [in]          const D3D11_INPUT_ELEMENT_DESC *pInputElementDescs,
    [in]          UINT                         NumElements,
    [in]          const void                    *pShaderBytecodeWithInputSignature,
    [in]          SIZE_T                      BytecodeLength,
    [out, optional] ID3D11InputLayout **ppInputLayout
);
```

## Parameters

[in] pInputElementDescs

Type: **const D3D11\_INPUT\_ELEMENT\_DESC\***

An array of the input-assembler stage input data types; each type is described by an element description (see [D3D11\\_INPUT\\_ELEMENT\\_DESC](#)).

[in] NumElements

Type: **UINT**

The number of input-data types in the array of input-elements.

[in] pShaderBytecodeWithInputSignature

Type: **const void\***

A pointer to the compiled shader. The compiled shader code contains a input signature which is validated against the array of elements. See remarks.

[in] BytecodeLength

Type: [SIZE\\_T](#)

Size of the compiled shader.

[out, optional] `ppInputLayout`

Type: [ID3D11InputLayout\\*\\*](#)

A pointer to the input-layout object created (see [ID3D11InputLayout](#)). To validate the other input parameters, set this pointer to be **NULL** and verify that the method returns **S\_FALSE**.

## Return value

Type: [HRESULT](#)

If the method succeeds, the return code is **S\_OK**. See [Direct3D 11 Return Codes](#) for failing error codes.

## Remarks

After creating an input layout object, it must be bound to the input-assembler stage before calling a draw API.

Once an input-layout object is created from a shader signature, the input-layout object can be reused with any other shader that has an identical input signature (semantics included). This can simplify the creation of input-layout objects when you are working with many shaders with identical inputs.

If a data type in the input-layout declaration does not match the data type in a shader-input signature, `CreateInputLayout` will generate a warning during compilation. The warning is simply to call attention to the fact that the data may be reinterpreted when read from a register. You may either disregard this warning (if reinterpretation is intentional) or make the data types match in both declarations to eliminate the warning.

**Windows Phone 8:** This API is supported.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreatePixelShader method (d3d11.h)

Article 02/22/2024

Create a pixel shader.

## Syntax

C++

```
HRESULT CreatePixelShader(  
    [in]          const void      *pShaderBytecode,  
    [in]          SIZE_T         BytecodeLength,  
    [in, optional] ID3D11ClassLinkage *pClassLinkage,  
    [out, optional] ID3D11PixelShader **ppPixelShader  
) ;
```

## Parameters

[in] pShaderBytecode

Type: **const void\***

A pointer to the compiled shader.

[in] BytecodeLength

Type: **SIZE\_T**

Size of the compiled pixel shader.

[in, optional] pClassLinkage

Type: **ID3D11ClassLinkage\***

A pointer to a class linkage interface (see [ID3D11ClassLinkage](#)); the value can be **NULL**.

[out, optional] ppPixelShader

Type: **ID3D11PixelShader\*\***

Address of a pointer to a [ID3D11PixelShader](#) interface. If this is **NULL**, all other parameters will be validated, and if all parameters pass validation this API will return

S\_FALSE instead of S\_OK.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

After creating the pixel shader, you can set it to the device using [ID3D11DeviceContext::PSSetShader](#).

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreatePredicate method (d3d11.h)

Article 02/22/2024

Creates a predicate.

## Syntax

C++

```
HRESULT CreatePredicate(
    [in]           const D3D11_QUERY_DESC *pPredicateDesc,
    [out, optional] ID3D11Predicate     **ppPredicate
);
```

## Parameters

[in] pPredicateDesc

Type: [const D3D11\\_QUERY\\_DESC\\*](#)

Pointer to a query description where the type of query must be a D3D11\_QUERY\_SO\_OVERFLOW\_PREDICATE or D3D11\_QUERY\_OCCLUSION\_PREDICATE (see [D3D11\\_QUERY\\_DESC](#)).

[out, optional] ppPredicate

Type: [ID3D11Predicate\\*\\*](#)

Address of a pointer to a predicate (see [ID3D11Predicate](#)).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateQuery method (d3d11.h)

Article 02/22/2024

This interface encapsulates methods for querying information from the GPU.

## Syntax

C++

```
HRESULT CreateQuery(  
    [in]           const D3D11_QUERY_DESC *pQueryDesc,  
    [out, optional] ID3D11Query        **ppQuery  
) ;
```

## Parameters

[in] pQueryDesc

Type: [const D3D11\\_QUERY\\_DESC\\*](#)

Pointer to a query description (see [D3D11\\_QUERY\\_DESC](#)).

[out, optional] ppQuery

Type: [ID3D11Query\\*\\*](#)

Address of a pointer to the query object created (see [ID3D11Query](#)).

## Return value

Type: [HRESULT](#)

This method returns [E\\_OUTOFMEMORY](#) if there is insufficient memory to create the query object.

See [Direct3D 11 Return Codes](#) for other possible return values.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateRasterizerState method (d3d11.h)

Article 02/22/2024

Create a rasterizer state object that tells the rasterizer stage how to behave.

## Syntax

C++

```
HRESULT CreateRasterizerState(  
    [in]          const D3D11_RASTERIZER_DESC *pRasterizerDesc,  
    [out, optional] ID3D11RasterizerState     **ppRasterizerState  
) ;
```

## Parameters

[in] pRasterizerDesc

Type: [const D3D11\\_RASTERIZER\\_DESC\\*](#)

Pointer to a rasterizer state description (see [D3D11\\_RASTERIZER\\_DESC](#)).

[out, optional] ppRasterizerState

Type: [ID3D11RasterizerState\\*\\*](#)

Address of a pointer to the rasterizer state object created (see [ID3D11RasterizerState](#)).

## Return value

Type: [HRESULT](#)

This method returns [E\\_OUTOFMEMORY](#) if there is insufficient memory to create the compute shader. See [Direct3D 11 Return Codes](#) for other possible return values.

## Remarks

4096 unique rasterizer state objects can be created on a device at a time.

If an application attempts to create a rasterizer-state interface with the same state as an existing interface, the same interface will be returned and the total number of unique rasterizer state objects will stay the same.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateRenderTargetView method (d3d11.h)

Article 02/22/2024

Creates a render-target view for accessing resource data.

## Syntax

C++

```
HRESULT CreateRenderTargetView(  
    [in]          ID3D11Resource           *pResource,  
    [in, optional] const D3D11_RENDER_TARGET_VIEW_DESC *pDesc,  
    [out, optional] ID3D11RenderTargetView        **ppRTView  
) ;
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

Pointer to a [ID3D11Resource](#) that represents a render target. This resource must have been created with the [D3D11\\_BIND\\_RENDER\\_TARGET](#) flag.

[in, optional] pDesc

Type: [const D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC\\*](#)

Pointer to a [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) that represents a render-target view description. Set this parameter to **NULL** to create a view that accesses all of the subresources in mipmap level 0.

[out, optional] ppRTView

Type: [ID3D11RenderTargetView\\*\\*](#)

Address of a pointer to an [ID3D11RenderTargetView](#). Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

# Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

A render-target view can be bound to the output-merger stage by calling [ID3D11DeviceContext::OMSetRenderTargets](#).

The Direct3D 11.1 runtime, which is available starting with Windows 8, allows you to use [CreateRenderTargetView](#) for the following new purpose.

You can create render-target views of video resources so that Direct3D shaders can process those render-target views. These video resources are either [Texture2D](#) or [Texture2DArray](#). The value in the [ViewDimension](#) member of the [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure for a created render-target view must match the type of video resource, [D3D11\\_RT\\_V\\_DIMENSION\\_TEXTURE2D](#) for Texture2D and [D3D11\\_RT\\_V\\_DIMENSION\\_TEXTURE2DARRAY](#) for Texture2DArray. Additionally, the format of the underlying video resource restricts the formats that the view can use. The video resource format values on the [DXGI\\_FORMAT](#) reference page specify the format values that views are restricted to.

The runtime read+write conflict prevention logic (which stops a resource from being bound as an SRV and RTV or UAV at the same time) treats views of different parts of the same video surface as conflicting for simplicity. Therefore, the runtime does not allow an application to read from luma while the application simultaneously renders to chroma in the same surface even though the hardware might allow these simultaneous operations.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateSamplerState method (d3d11.h)

Article 02/22/2024

Create a sampler-state object that encapsulates sampling information for a texture.

## Syntax

C++

```
HRESULT CreateSamplerState(
    [in]          const D3D11_SAMPLER_DESC *pSamplerDesc,
    [out, optional] ID3D11SamplerState     **ppSamplerState
);
```

## Parameters

[in] pSamplerDesc

Type: [const D3D11\\_SAMPLER\\_DESC\\*](#)

Pointer to a sampler state description (see [D3D11\\_SAMPLER\\_DESC](#)).

[out, optional] ppSamplerState

Type: [ID3D11SamplerState\\*\\*](#)

Address of a pointer to the sampler state object created (see [ID3D11SamplerState](#)).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

4096 unique sampler state objects can be created on a device at a time.

If an application attempts to create a sampler-state interface with the same state as an existing interface, the same interface will be returned and the total number of unique sampler state objects will stay the same.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateShaderResourceView method (d3d11.h)

Article 02/22/2024

Create a shader-resource view for accessing data in a resource.

## Syntax

C++

```
HRESULT CreateShaderResourceView(
    [in]          ID3D11Resource                  *pResource,
    [in, optional] const D3D11_SHADER_RESOURCE_VIEW_DESC *pDesc,
    [out, optional] ID3D11ShaderResourceView           **ppSRView
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

Pointer to the resource that will serve as input to a shader. This resource must have been created with the [D3D11\\_BIND\\_SHADER\\_RESOURCE](#) flag.

[in, optional] pDesc

Type: [const D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC\\*](#)

Pointer to a shader-resource view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)). Set this parameter to **NULL** to create a view that accesses the entire resource (using the format the resource was created with).

[out, optional] ppSRView

Type: [ID3D11ShaderResourceView\\*\\*](#)

Address of a pointer to an [ID3D11ShaderResourceView](#). Set this parameter to **NULL** to validate the other input parameters (the method will return [S\\_FALSE](#) if the other input parameters pass validation).

# Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

A resource is made up of one or more subresources; a view identifies which subresources to allow the pipeline to access. In addition, each resource is bound to the pipeline using a view. A shader-resource view is designed to bind any buffer or texture resource to the shader stages using the following API methods:

[ID3D11DeviceContext::VSSetShaderResources](#),  
[ID3D11DeviceContext::GSSetShaderResources](#) and  
[ID3D11DeviceContext::PSSetShaderResources](#).

Because a view is fully typed, this means that typeless resources become fully typed when bound to the pipeline.

**Note** To successfully create a shader-resource view from a typeless buffer (for example, [DXGI FORMAT R32G32B32A32 TYPELESS](#)), you must set the [D3D11 RESOURCE\\_MISC\\_BUFFER\\_ALLOW\\_RAW\\_VIEWS](#) flag when you create the buffer.

The Direct3D 11.1 runtime, which is available starting with Windows 8, allows you to use [CreateShaderResourceView](#) for the following new purpose.

You can create shader-resource views of video resources so that Direct3D shaders can process those shader-resource views. These video resources are either [Texture2D](#) or [Texture2DArray](#). The value in the [ViewDimension](#) member of the [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure for a created shader-resource view must match the type of video resource, [D3D11\\_SRV\\_DIMENSION\\_TEXTURE2D](#) for [Texture2D](#) and [D3D11\\_SRV\\_DIMENSION\\_TEXTURE2DARRAY](#) for [Texture2DArray](#). Additionally, the format of the underlying video resource restricts the formats that the view can use. The video resource format values on the [DXGI\\_FORMAT](#) reference page specify the format values that views are restricted to.

The runtime read+write conflict prevention logic (which stops a resource from being bound as an SRV and RTV or UAV at the same time) treats views of different parts of the same video surface as conflicting for simplicity. Therefore, the runtime does not allow an

application to read from luma while the application simultaneously renders to chroma in the same surface even though the hardware might allow these simultaneous operations.

**Windows Phone 8:** This API is supported.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateTexture1D method (d3d11.h)

Article 10/13/2021

Creates an array of 1D textures.

## Syntax

C++

```
HRESULT CreateTexture1D(  
    [in]          const D3D11_TEXTURE1D_DESC  *pDesc,  
    [in, optional] const D3D11_SUBRESOURCE_DATA *pInitialData,  
    [out, optional] ID3D11Texture1D           **ppTexture1D  
) ;
```

## Parameters

[in] pDesc

Type: [const D3D11\\_TEXTURE1D\\_DESC\\*](#)

A pointer to a [D3D11\\_TEXTURE1D\\_DESC](#) structure that describes a 1D texture resource. To create a typeless resource that can be interpreted at runtime into different, compatible formats, specify a typeless format in the texture description. To generate mipmap levels automatically, set the number of mipmap levels to 0.

[in, optional] pInitialData

Type: [const D3D11\\_SUBRESOURCE\\_DATA\\*](#)

A pointer to an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures that describe subresources for the 1D texture resource. Applications can't specify **NULL** for *pInitialData* when creating IMMUTABLE resources (see [D3D11\\_USAGE](#)). If the resource is multisampled, *pInitialData* must be **NULL** because multisampled resources cannot be initialized with data when they are created.

If you don't pass anything to *pInitialData*, the initial content of the memory for the resource is undefined. In this case, you need to write the resource content some other way before the resource is read.

You can determine the size of this array from values in the **MipLevels** and **ArraySize** members of the [D3D11\\_TEXTURE1D\\_DESC](#) structure to which *pDesc* points by using the following calculation:

$$\text{MipLevels} * \text{ArraySize}$$

For more information about this array size, see Remarks.

[out, optional] *ppTexture1D*

Type: [ID3D11Texture1D\\*\\*](#)

A pointer to a buffer that receives a pointer to a [ID3D11Texture1D](#) interface for the created texture. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

## Return value

Type: [HRESULT](#)

If the method succeeds, the return code is **S\_OK**. See [Direct3D 11 Return Codes](#) for failing error codes.

## Remarks

[CreateTexture1D](#) creates a 1D texture resource, which can contain a number of 1D subresources. The number of textures is specified in the texture description. All textures in a resource must have the same format, size, and number of mipmap levels.

All resources are made up of one or more subresources. To load data into the texture, applications can supply the data initially as an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures pointed to by *pInitialData*, or they can use one of the D3DX texture functions such as [D3DX11CreateTextureFromFile](#).

For a 32 width texture with a full mipmap chain, the *pInitialData* array has the following 6 elements:

- *pInitialData[0]* = 32x1
- *pInitialData[1]* = 16x1
- *pInitialData[2]* = 8x1
- *pInitialData[3]* = 4x1
- *pInitialData[4]* = 2x1
- *pInitialData[5]* = 1x1

# Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateTexture2D method (d3d11.h)

Article 10/13/2021

Create an array of [2D textures](#).

## Syntax

C++

```
HRESULT CreateTexture2D(  
    [in]          const D3D11_TEXTURE2D_DESC  *pDesc,  
    [in, optional] const D3D11_SUBRESOURCE_DATA *pInitialData,  
    [out, optional] ID3D11Texture2D           **ppTexture2D  
) ;
```

## Parameters

[in] pDesc

Type: [const D3D11\\_TEXTURE2D\\_DESC\\*](#)

A pointer to a [D3D11\\_TEXTURE2D\\_DESC](#) structure that describes a 2D texture resource. To create a typeless resource that can be interpreted at runtime into different, compatible formats, specify a typeless format in the texture description. To generate mipmap levels automatically, set the number of mipmap levels to 0.

[in, optional] pInitialData

Type: [const D3D11\\_SUBRESOURCE\\_DATA\\*](#)

A pointer to an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures that describe subresources for the 2D texture resource. Applications can't specify **NULL** for *pInitialData* when creating IMMUTABLE resources (see [D3D11\\_USAGE](#)). If the resource is multisampled, *pInitialData* must be **NULL** because multisampled resources cannot be initialized with data when they are created.

If you don't pass anything to *pInitialData*, the initial content of the memory for the resource is undefined. In this case, you need to write the resource content some other way before the resource is read.

You can determine the size of this array from values in the **MipLevels** and **ArraySize** members of the [D3D11\\_TEXTURE2D\\_DESC](#) structure to which *pDesc* points by using the following calculation:

$$\text{MipLevels} * \text{ArraySize}$$

For more information about this array size, see Remarks.

`[out, optional] ppTexture2D`

Type: [ID3D11Texture2D\\*\\*](#)

A pointer to a buffer that receives a pointer to a [ID3D11Texture2D](#) interface for the created texture. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

## Return value

Type: [HRESULT](#)

If the method succeeds, the return code is **S\_OK**. See [Direct3D 11 Return Codes](#) for failing error codes.

## Remarks

[CreateTexture2D](#) creates a 2D texture resource, which can contain a number of 2D subresources. The number of textures is specified in the texture description. All textures in a resource must have the same format, size, and number of mipmap levels.

All resources are made up of one or more subresources. To load data into the texture, applications can supply the data initially as an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures pointed to by *pInitialData*, or it may use one of the D3DX texture functions such as [D3DX11CreateTextureFromFile](#).

For a 32 x 32 texture with a full mipmap chain, the *pInitialData* array has the following 6 elements:

- *pInitialData[0]* = 32x32
- *pInitialData[1]* = 16x16
- *pInitialData[2]* = 8x8
- *pInitialData[3]* = 4x4
- *pInitialData[4]* = 2x2
- *pInitialData[5]* = 1x1

# Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateTexture3D method (d3d11.h)

Article 10/13/2021

Create a single [3D texture](#).

## Syntax

C++

```
HRESULT CreateTexture3D(  
    [in]          const D3D11_TEXTURE3D_DESC  *pDesc,  
    [in, optional] const D3D11_SUBRESOURCE_DATA *pInitialData,  
    [out, optional] ID3D11Texture3D           **ppTexture3D  
) ;
```

## Parameters

[in] pDesc

Type: [const D3D11\\_TEXTURE3D\\_DESC\\*](#)

A pointer to a [D3D11\\_TEXTURE3D\\_DESC](#) structure that describes a 3D texture resource. To create a typeless resource that can be interpreted at runtime into different, compatible formats, specify a typeless format in the texture description. To generate mipmap levels automatically, set the number of mipmap levels to 0.

[in, optional] pInitialData

Type: [const D3D11\\_SUBRESOURCE\\_DATA\\*](#)

A pointer to an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures that describe subresources for the 3D texture resource. Applications cannot specify **NULL** for *pInitialData* when creating IMMUTABLE resources (see [D3D11\\_USAGE](#)). If the resource is multisampled, *pInitialData* must be **NULL** because multisampled resources cannot be initialized with data when they are created.

If you don't pass anything to *pInitialData*, the initial content of the memory for the resource is undefined. In this case, you need to write the resource content some other way before the resource is read.

You can determine the size of this array from the value in the **MipLevels** member of the [D3D11\\_TEXTURE3D\\_DESC](#) structure to which *pDesc* points. Arrays of 3D volume textures are not supported.

For more information about this array size, see Remarks.

[out, optional] *ppTexture3D*

Type: [ID3D11Texture3D\\*\\*](#)

A pointer to a buffer that receives a pointer to a [ID3D11Texture3D](#) interface for the created texture. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

## Return value

Type: [HRESULT](#)

If the method succeeds, the return code is **S\_OK**. See [Direct3D 11 Return Codes](#) for failing error codes.

## Remarks

`CreateTexture3D` creates a 3D texture resource, which can contain a number of 3D subresources. The number of textures is specified in the texture description. All textures in a resource must have the same format, size, and number of mipmap levels.

All resources are made up of one or more subresources. To load data into the texture, applications can supply the data initially as an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures pointed to by *pInitialData*, or they can use one of the D3DX texture functions such as [D3DX11CreateTextureFromFile](#).

Each element of *pInitialData* provides all of the slices that are defined for a given mipmap level. For example, for a 32 x 32 x 4 volume texture with a full mipmap chain, the array has the following 6 elements:

- *pInitialData[0]* = 32x32 with 4 slices
- *pInitialData[1]* = 16x16 with 2 slices
- *pInitialData[2]* = 8x8 with 1 slice
- *pInitialData[3]* = 4x4 with 1 slice
- *pInitialData[4]* = 2x2 with 1 slice
- *pInitialData[5]* = 1x1 with 1 slice

# Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateUnorderedAccessView method (d3d11.h)

Article 07/27/2022

Creates a view for accessing an [unordered access](#) resource.

## Syntax

C++

```
HRESULT CreateUnorderedAccessView(
    [in]           ID3D11Resource                  *pResource,
    [in, optional] const D3D11_UNORDERED_ACCESS_VIEW_DESC *pDesc,
    [out, optional] ID3D11UnorderedAccessView          **ppUAVview
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource](#)\*

Pointer to an [ID3D11Resource](#) that represents a resources that will serve as an input to a shader.

[in, optional] pDesc

Type: [const D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)\*

Pointer to an [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) that represents a shader-resource view description. Set this parameter to **NULL** to create a view that accesses the entire resource (using the format the resource was created with).

[out, optional] ppUAVview

Type: [ID3D11UnorderedAccessView](#)\*\*

Address of a pointer to an [ID3D11UnorderedAccessView](#) that represents an unordered-access view. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

# Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

The Direct3D 11.1 runtime, which is available starting with Windows 8, allows you to use [CreateUnorderedAccessView](#) for the following new purpose.

You can create unordered-access views of video resources so that Direct3D shaders can process those unordered-access views. These video resources are either [Texture2D](#) or [Texture2DArray](#). The value in the [ViewDimension](#) member of the [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure for a created unordered-access view must match the type of video resource, [D3D11\\_UAV\\_DIMENSION\\_TEXTURE2D](#) for [Texture2D](#) and [D3D11\\_UAV\\_DIMENSION\\_TEXTURE2DARRAY](#) for [Texture2DArray](#). Additionally, the format of the underlying video resource restricts the formats that the view can use. The video resource format values on the [DXGI\\_FORMAT](#) reference page specify the format values that views are restricted to.

The runtime read+write conflict prevention logic (which stops a resource from being bound as an SRV and RTV or UAV at the same time) treats views of different parts of the same video surface as conflicting for simplicity. Therefore, the runtime does not allow an application to read from luma while the application simultaneously renders to chroma in the same surface even though the hardware might allow these simultaneous operations.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::CreateVertexShader method (d3d11.h)

Article 02/22/2024

Create a vertex-shader object from a compiled shader.

## Syntax

C++

```
HRESULT CreateVertexShader(  
    [in]          const void      *pShaderBytecode,  
    [in]          SIZE_T          BytecodeLength,  
    [in, optional] ID3D11ClassLinkage *pClassLinkage,  
    [out, optional] ID3D11VertexShader **ppVertexShader  
) ;
```

## Parameters

[in] pShaderBytecode

Type: **const void\***

A pointer to the compiled shader.

[in] BytecodeLength

Type: **SIZE\_T**

Size of the compiled vertex shader.

[in, optional] pClassLinkage

Type: **ID3D11ClassLinkage\***

A pointer to a class linkage interface (see [ID3D11ClassLinkage](#)); the value can be **NULL**.

[out, optional] ppVertexShader

Type: **ID3D11VertexShader\*\***

Address of a pointer to a [ID3D11VertexShader](#) interface. If this is **NULL**, all other parameters will be validated, and if all parameters pass validation this API will return

`S_FALSE` instead of `S_OK`.

## Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

The Direct3D 11.1 runtime, which is available starting with Windows 8, provides the following new functionality for [CreateVertexShader](#).

The following shader model 5.0 instructions are available to just pixel shaders and compute shaders in the Direct3D 11.0 runtime. For the Direct3D 11.1 runtime, because unordered access views (UAV) are available at all shader stages, you can use these instructions in all shader stages.

Therefore, if you use the following shader model 5.0 instructions in a vertex shader, you can successfully pass the compiled vertex shader to *pShaderBytecode*. That is, the call to [CreateVertexShader](#) succeeds.

If you pass a compiled shader to *pShaderBytecode* that uses any of the following instructions on a device that doesn't support UAVs at every shader stage (including existing drivers that are not implemented to support UAVs at every shader stage), [CreateVertexShader](#) fails. [CreateVertexShader](#) also fails if the shader tries to use a UAV slot beyond the set of UAV slots that the hardware supports.

- [dcl\\_uav\\_typed](#)
- [dcl\\_uav\\_raw](#)
- [dcl\\_uav\\_structured](#)
- [ld\\_raw](#)
- [ld\\_structured](#)
- [ld\\_uav\\_typed](#)
- [store\\_raw](#)
- [store\\_structured](#)
- [store\\_uav\\_typed](#)
- [sync\\_uglobal](#)
- All atomics and immediate atomics (for example, [atomic\\_and](#) and [imm\\_atomic\\_and](#))

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::GetCreationFlags method (d3d11.h)

Article 02/22/2024

Get the flags used during the call to create the device with [D3D11CreateDevice](#).

## Syntax

C++

```
UINT GetCreationFlags();
```

## Return value

Type: [UINT](#)

A bitfield containing the flags used to create the device. See [D3D11\\_CREATE\\_DEVICE\\_FLAG](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11Device::GetDeviceRemovedReason method (d3d11.h)

Article 02/22/2024

Get the reason why the device was removed.

## Syntax

C++

```
HRESULT GetDeviceRemovedReason();
```

## Return value

Type: [HRESULT](#)

Possible return values include:

- DXGI\_ERROR\_DEVICE\_HUNG
- DXGI\_ERROR\_DEVICE\_REMOVED
- DXGI\_ERROR\_DEVICE\_RESET
- DXGI\_ERROR\_DRIVER\_INTERNAL\_ERROR
- DXGI\_ERROR\_INVALID\_CALL
- S\_OK

For more detail on these return codes, see [DXGI\\_ERROR](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::GetExceptionMode method (d3d11.h)

Article 02/22/2024

Get the exception-mode flags.

## Syntax

C++

```
UINT GetExceptionMode();
```

## Return value

Type: [UINT](#)

A value that contains one or more exception flags; each flag specifies a condition which will cause an exception to be raised. The flags are listed in [D3D11\\_RAISE\\_FLAG](#). A default value of 0 means there are no flags.

## Remarks

An exception-mode flag is used to elevate an error condition to a non-continuable exception.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::GetFeatureLevel method (d3d11.h)

Article02/22/2024

Gets the feature level of the hardware device.

## Syntax

C++

```
D3D_FEATURE_LEVEL GetFeatureLevel();
```

## Return value

Type: [D3D\\_FEATURE\\_LEVEL](#)

A member of the [D3D\\_FEATURE\\_LEVEL](#) enumerated type that describes the feature level of the hardware device.

## Remarks

[Feature levels](#) determine the capabilities of your device.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::GetImmediateContext method (d3d11.h)

Article 02/22/2024

Gets an immediate context, which can play back command lists.

## Syntax

C++

```
void GetImmediateContext(  
    [out] ID3D11DeviceContext **ppImmediateContext  
);
```

## Parameters

[out] ppImmediateContext

Type: [ID3D11DeviceContext\\*\\*](#)

Upon completion of the method, the passed pointer to an [ID3D11DeviceContext](#) interface pointer is initialized.

## Return value

None

## Remarks

The **GetImmediateContext** method returns an [ID3D11DeviceContext](#) object that represents an immediate context which is used to perform rendering that you want immediately submitted to a device. For most applications, an immediate context is the primary object that is used to draw your scene.

The **GetImmediateContext** method increments the reference count of the immediate context by one. Therefore, you must call [Release](#) on the returned interface pointer when you are done with it to avoid a memory leak.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::GetPrivateData method (d3d11.h)

Article 02/22/2024

Get application-defined data from a device.

## Syntax

C++

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

## Parameters

[in] guid

Type: [REFGUID](#)

Guid associated with the data.

[in, out] pDataSize

Type: [UINT\\*](#)

A pointer to a variable that on input contains the size, in bytes, of the buffer that *pData* points to, and on output contains the size, in bytes, of the amount of data that *GetPrivateData* retrieved.

[out, optional] pData

Type: [void\\*](#)

A pointer to a buffer that *GetPrivateData* fills with data from the device 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 codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

If the data returned is a pointer to an [IUnknown](#), or one of its derivative classes, which was previously set by SetPrivateDataInterface, that interface will have its reference count incremented before the private data is returned.

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::OpenSharedResource method (d3d11.h)

Article 02/22/2024

Give a device access to a shared resource created on a different device.

## Syntax

C++

```
HRESULT OpenSharedResource(
    [in]          HANDLE hResource,
    [in]          REFIID ReturnedInterface,
    [out, optional] void   **ppResource
);
```

## Parameters

[in] hResource

Type: [HANDLE](#)

A resource handle. See remarks.

[in] ReturnedInterface

Type: [REFIID](#)

The globally unique identifier (GUID) for the resource interface. See remarks.

[out, optional] ppResource

Type: [void\\*\\*](#)

Address of a pointer to the resource we are gaining access to.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

The REFIID, or GUID, of the interface to the resource can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D11Buffer)` will get the GUID of the interface to a buffer resource.

The unique handle of the resource is obtained differently depending on the type of device that originally created the resource.

To share a resource between two Direct3D 11 devices the resource must have been created with the `D3D11_RESOURCE_MISC_SHARED` flag, if it was created using the `ID3D11Device` interface. If it was created using a DXGI device interface, then the resource is always shared.

The REFIID, or GUID, of the interface to the resource can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D11Buffer)` will get the GUID of the interface to a buffer resource.

When sharing a resource between two Direct3D 10/11 devices the unique handle of the resource can be obtained by querying the resource for the `IDXGIResource` interface and then calling `GetSharedHandle`.

```
IDXGIResource* p0OtherResource(NULL);
hr = p0OtherDeviceResource->QueryInterface( __uuidof(IDXGIResource),
(void**)&p0OtherResource );
HANDLE sharedHandle;
p0OtherResource->GetSharedHandle(&sharedHandle);
```

The only resources that can be shared are 2D non-mipmapped textures.

To share a resource between a Direct3D 9 device and a Direct3D 11 device the texture must have been created using the `pSharedHandle` argument of `CreateTexture`.

The shared Direct3D 9 handle is then passed to `OpenSharedResource` in the `hResource` argument.

The following code illustrates the method calls involved.

```
sharedHandle = NULL; // must be set to NULL to create, can use a valid
handle here to open in D3D9
```

```

pDevice9->CreateTexture(..., pTex2D_9, &sharedHandle);
...
pDevice11->OpenSharedResource(sharedHandle, __uuidof(ID3D11Resource),
(void**)(&tempResource11));
tempResource11->QueryInterface(__uuidof(ID3D11Texture2D), (void**)
(&pTex2D_11));
tempResource11->Release();
// now use pTex2D_11 with pDevice11

```

Textures being shared from D3D9 to D3D11 have the following restrictions.

- Textures must be 2D
- Only 1 mip level is allowed
- Texture must have default usage
- Texture must be write only
- MSAA textures are not allowed
- Bind flags must have SHADER\_RESOURCE and RENDER\_TARGET set
- Only R10G10B10A2\_UNORM, R16G16B16A16\_FLOAT and R8G8B8A8\_UNORM formats are allowed

If a shared texture is updated on one device [ID3D11DeviceContext::Flush](#) must be called on that device.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11Device::SetExceptionMode method (d3d11.h)

Article 02/22/2024

Get the exception-mode flags.

## Syntax

C++

```
HRESULT SetExceptionMode(  
    UINT RaiseFlags  
);
```

## Parameters

RaiseFlags

Type: [UINT](#)

A value that contains one or more exception flags; each flag specifies a condition which will cause an exception to be raised. The flags are listed in [D3D11\\_RAISE\\_FLAG](#). A default value of 0 means there are no flags.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Set an exception-mode flag to elevate an error condition to a non-continuable exception.

Whenever an error occurs, a Direct3D device enters the DEVICEREMOVED state and if the appropriate exception flag has been set, an exception is raised. A raised exception is designed to terminate an application. Before termination, the last chance an application has to persist data is by using an UnhandledExceptionFilter (see [Structured Exception](#)

[Handling](#)). In general, UnhandledExceptionFilters are leveraged to try to persist data when an application is crashing (to disk, for example). Any code that executes during an UnhandledExceptionFilter is not guaranteed to reliably execute (due to possible process corruption). Any data that the UnhandledExceptionFilter manages to persist, before the UnhandledExceptionFilter crashes again, should be treated as suspect, and therefore inspected by a new, non-corrupted process to see if it is usable.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device::SetPrivateData method (d3d11.h)

Article 02/22/2024

Set data to a device and associate that data with a guid.

## Syntax

C++

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

## Parameters

[in] guid

Type: [REFGUID](#)

Guid associated with the data.

[in] DataSize

Type: [UINT](#)

Size of the data.

[in, optional] pData

Type: [const void\\*](#)

Pointer to the data to be stored with this device. If pData is **NULL**, DataSize must also be 0, and any data previously associated with the guid will be destroyed.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

The data stored in the device with this method can be retrieved with [ID3D11Device::GetPrivateData](#).

The data and guid set with this method will typically be application-defined.

The [debug layer](#) reports memory leaks by outputting a list of object interface pointers along with their friendly names. The default friendly name is "<unnamed>". You can set the friendly name so that you can determine if the corresponding object interface pointer caused the leak. To set the friendly name, use the **SetPrivateData** method and the **WKPID\_D3DDescribeObjectName** GUID that is in D3Dcommon.h. For example, to give pContext a friendly name of *My name*, use the following code:

```
static const char c_szName[] = "My name";
hr = pContext->SetPrivateData( WKPID_D3DDescribeObjectName, sizeof( c_szName )
- 1, c_szName );
```

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

## Feedback

Was this page helpful?

 Yes

 No



# ID3D11Device::SetPrivateDataInterface method (d3d11.h)

Article 02/22/2024

Associate an IUnknown-derived interface with this device child and associate that interface with an application-defined guid.

## Syntax

C++

```
HRESULT SetPrivateDataInterface(
    [in]           REFGUID      guid,
    [in, optional] const IUnknown *pData
);
```

## Parameters

[in] guid

Type: [REFGUID](#)

Guid associated with the interface.

[in, optional] pData

Type: [const IUnknown\\*](#)

Pointer to an IUnknown-derived interface to be associated with the device child.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Device](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device1 interface (d3d11\_1.h)

Article02/22/2024

The device interface represents a virtual adapter; it is used to create resources. ID3D11Device1 adds new methods to those in [ID3D11Device](#).

## Inheritance

The **ID3D11Device1** interface inherits from [ID3D11Device](#). **ID3D11Device1** also has these types of members:

## Methods

The **ID3D11Device1** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11Device1::CreateBlendState1</a>
Creates a blend-state object that encapsulates blend state for the output-merger stage and allows the configuration of logic operations.
<a href="#">ID3D11Device1::CreateDeferredContext1</a>
Creates a deferred context, which can record command lists. ( <code>ID3D11Device1.CreateDeferredContext1</code> )
<a href="#">ID3D11Device1::CreateDeviceContextState</a>
Creates a context state object that holds all Microsoft Direct3D state and some Direct3D behavior.
<a href="#">ID3D11Device1::CreateRasterizerState1</a>
Creates a rasterizer state object that informs the rasterizer stage how to behave and forces the sample count while UAV rendering or rasterizing. ( <code>ID3D11Device1.CreateRasterizerState1</code> )
<a href="#">ID3D11Device1::GetImmediateContext1</a>
Gets an immediate context, which can play back command lists. ( <code>ID3D11Device1.GetImmediateContext1</code> )
<a href="#">ID3D11Device1::OpenSharedResource1</a>

Gives a device access to a shared resource that is referenced by a handle and that was created on a different device.

#### [ID3D11Device1::OpenSharedResourceByName](#)

Gives a device access to a shared resource that is referenced by name and that was created on a different device.

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h

## See also

[Core Interfaces](#)

[ID3D11Device](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device1::CreateBlendState1 method (d3d11\_1.h)

Article 02/22/2024

Creates a blend-state object that encapsulates blend state for the [output-merger stage](#) and allows the configuration of logic operations.

## Syntax

C++

```
HRESULT CreateBlendState1(
    [in]           const D3D11_BLEND_DESC1 *pBlendStateDesc,
    [out, optional] ID3D11BlendState1     **ppBlendState
);
```

## Parameters

[in] pBlendStateDesc

A pointer to a [D3D11\\_BLEND\\_DESC1](#) structure that describes blend state.

[out, optional] ppBlendState

Address of a pointer to the [ID3D11BlendState1](#) interface for the blend-state object created.

## Return value

This method returns [E\\_OUTOFMEMORY](#) if there is insufficient memory to create the blend-state object.

See [Direct3D 11 Return Codes](#) for other possible return values.

## Remarks

The logical operations (those that enable bitwise logical operations between pixel shader output and render target contents, refer to

[D3D11\\_RENDER\\_TARGET\\_BLEND\\_DESC1](#) ) are only available on certain feature levels; call [CheckFeatureSupport](#) with [D3D11\\_FEATURE\\_D3D11\\_OPTIONS](#) set, to ensure support by

checking the boolean field *OutputMergerLogicOp* of [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#).

An app can create up to 4096 unique blend-state objects. For each object created, the runtime checks to see if a previous object has the same state. If such a previous object exists, the runtime will return a pointer to previous instance instead of creating a duplicate object.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11Device1](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device1::CreateDeferredContext1 method (d3d11\_1.h)

Article 07/27/2022

Creates a deferred context, which can record command lists.

## Syntax

C++

```
HRESULT CreateDeferredContext1(
    UINT             ContextFlags,
    [out, optional] ID3D11DeviceContext1 **ppDeferredContext
);
```

## Parameters

ContextFlags

Reserved for future use. Pass 0.

[out, optional] ppDeferredContext

Upon completion of the method, the passed pointer to an [ID3D11DeviceContext1](#) interface pointer is initialized.

## Return value

Returns S\_OK if successful; otherwise, returns one of the following:

- Returns DXGI\_ERROR\_DEVICE\_REMOVED if the graphics adapter has been physically removed from the computer or a driver upgrade for the graphics adapter has occurred. If this error occurs, you should destroy and re-create the device.
- Returns DXGI\_ERROR\_INVALID\_CALL if the CreateDeferredContext1 method cannot be called from the current context. For example, if the device was created with the [D3D11\\_CREATE\\_DEVICE\\_SINGLETHREADED](#) value, CreateDeferredContext1 returns DXGI\_ERROR\_INVALID\_CALL.
- Returns E\_INVALIDARG if the *ContextFlags* parameter is invalid.
- Returns E\_OUTOFMEMORY if the application has exhausted available memory.

## Remarks

A deferred context is a thread-safe context that you can use to record graphics commands on a thread other than the main rendering thread. By using a deferred context, you can record graphics commands into a command list that is encapsulated by the [ID3D11CommandList](#) interface. After you record all scene items, you can then submit them to the main render thread for final rendering. In this manner, you can perform rendering tasks concurrently across multiple threads and potentially improve performance in multi-core CPU scenarios.

You can create multiple deferred contexts.

**Note** If you use the `D3D11_CREATE_DEVICE_SINGLETHREADED` value to create the device that is represented by `ID3D11Device1`, the `CreateDeferredContext1` method will fail, and you will not be able to create a deferred context.

For more information about deferred contexts, see [Immediate and Deferred Rendering](#).

**Windows Phone 8:** This API is supported.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11Device1](#)

[ID3D11Device2::CreateDeferredContext2](#)

[ID3D11Device3::CreateDeferredContext3](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3D11Device1::CreateDeviceContextState method (d3d11\_1.h)

Article 10/13/2021

Creates a context state object that holds all Microsoft Direct3D state and some Direct3D behavior.

## Syntax

C++

```
HRESULT CreateDeviceContextState(
    [in]           UINT             Flags,
    [in]           const D3D_FEATURE_LEVEL *pFeatureLevels,
    [in]           UINT             FeatureLevels,
    [in]           UINT             SDKVersion,
    [in]           REFIID            EmulatedInterface,
    [out, optional] D3D_FEATURE_LEVEL *pChosenFeatureLevel,
    [out, optional] ID3DDeviceContextState **ppContextState
);
```

## Parameters

Flags

Type: [UINT](#)

A combination of [D3D11\\_1\\_CREATE\\_DEVICE\\_CONTEXT\\_STATE\\_FLAG](#) values that are combined by using a bitwise **OR** operation. The resulting value specifies how to create the context state object. The

[D3D11\\_1\\_CREATE\\_DEVICE\\_CONTEXT\\_STATE\\_SINGLETHREADED](#) flag is currently the only defined flag. If the original device was created with [D3D11\\_CREATE\\_DEVICE\\_SINGLETHREADED](#), you must create all context state objects from that device with the [D3D11\\_1\\_CREATE\\_DEVICE\\_CONTEXT\\_STATE\\_SINGLETHREADED](#) flag.

If you set the single-threaded flag for both the context state object and the device, you guarantee that you will call the whole set of context methods and device methods only from one thread. You therefore do not need to use critical sections to synchronize access to the device context, and the runtime can avoid working with those processor-intensive critical sections.

[in] pFeatureLevels

Type: **const D3D\_FEATURE\_LEVEL\***

A pointer to an array of **D3D\_FEATURE\_LEVEL** values. The array can contain elements from the following list and determines the order of feature levels for which creation is attempted. Unlike [D3D11CreateDevice](#), you can't set *pFeatureLevels* to **NULL** because there is no default feature level array.

```
{  
    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,  
};
```

FeatureLevels

Type: **UINT**

The number of elements in *pFeatureLevels*. Unlike [D3D11CreateDevice](#), you must set *FeatureLevels* to greater than 0 because you can't set *pFeatureLevels* to **NULL**.

SDKVersion

Type: **UINT**

The SDK version. You must set this parameter to **D3D11\_SDK\_VERSION**.

EmulatedInterface

Type: **REFIID**

The globally unique identifier (GUID) for the emulated interface. This value specifies the behavior of the device when the context state object is active. Valid values are obtained by using the **\_uuidof** operator on the [ID3D10Device](#), [ID3D10Device1](#), [ID3D11Device](#), and [ID3D11Device1](#) interfaces. See Remarks.

[out, optional] pChosenFeatureLevel

Type: **D3D\_FEATURE\_LEVEL\***

A pointer to a variable that receives a [D3D\\_FEATURE\\_LEVEL](#) value from the *pFeatureLevels* array. This is the first array value with which [CreateDeviceContextState](#) succeeded in creating the context state object. If the call to [CreateDeviceContextState](#) fails, the variable pointed to by *pChosenFeatureLevel* is set to zero.

[out, optional] *ppContextState*

Type: [ID3DDeviceContextState\\*\\*](#)

The address of a pointer to an [ID3DDeviceContextState](#) object that represents the state of a Direct3D device.

## Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

The **REFIID** value of the emulated interface is a GUID obtained by use of the `_uuidof` operator. For example, `_uuidof(ID3D11Device)` gets the GUID of the interface to a Microsoft Direct3D 11 device.

Call the [ID3D11DeviceContext1::SwapDeviceContextState](#) method to activate the context state object. When the context state object is active, the device behaviors that are associated with both the context state object's feature level and its compatible interface are activated on the Direct3D device until the next call to [SwapDeviceContextState](#).

When a context state object is active, the runtime disables certain methods on the device and context interfaces. For example, a context state object that is created with `_uuidof(ID3D11Device)` will cause the runtime to turn off most of the Microsoft Direct3D 10 device interfaces, and a context state object that is created with `_uuidof(ID3D10Device1)` or `_uuidof(ID3D10Device)` will cause the runtime to turn off most of the [ID3D11DeviceContext](#) methods. This behavior ensures that a user of either emulated interface cannot set device state that the other emulated interface is unable to express. This restriction helps guarantee that the [ID3D10Device1](#) emulated interface accurately reflects the full state of the pipeline and that the emulated interface will not operate contrary to its original interface definition.

For example, suppose the tessellation stage is made active through the [ID3D11DeviceContext](#) interface when you create the device through [D3D11CreateDevice](#)

or [D3D11CreateDeviceAndSwapChain](#), instead of through the Direct3D 10 equivalents. Because the Direct3D 11 context is active, a Direct3D 10 interface is inactive when you first retrieve it via [QueryInterface](#). This means that you cannot immediately pass a Direct3D 10 interface that you retrieved from a Direct3D 11 device to a function. You must first call [SwapDeviceContextState](#) to activate a Direct3D 10-compatible context state object.

The following table shows the methods that are active and inactive for each emulated interface.

<b>Emulated interface</b>	<b>Active device or immediate context interfaces</b>	<b>Inactive device or immediate context interfaces</b>
ID3D11Device or ID3D11Device1	ID3D11Device IDXGIDevice + IDXGIDevice1 + IDXGIDevice2 ID3D10Multithread	ID3D10Device
ID3D10Device1 or ID3D10Device	ID3D10Device ID3D10Device1 IDXGIDevice + IDXGIDevice1 ID3D10Multithread	ID3D11Device  ID3D11DeviceContext (As published by the immediate context. The Direct3D 10 or Microsoft Direct3D 10.1 emulated interface has no effect on deferred contexts.)

The following table shows the immediate context methods that the runtime disables when the indicated context state objects are active.

<b>Methods of ID3D11DeviceContext when __uuidof(ID3D10Device1) or __uuidof(ID3D10Device) is active</b>	<b>Methods of ID3D10Device when __uuidof(ID3D11Device) is active</b>
ClearDepthStencilView	ClearDepthStencilView
ClearRenderTargetView	ClearRenderTargetView
ClearState	ClearState
ClearUnorderedAccessViewUint	

ClearUnorderedAccessViewFloat	
CopyResource	CopyResource
CopyStructureCount	
CopySubresourceRegion	CopySubresourceRegion
CSGetConstantBuffers	
CSGetSamplers	
CSGetShader	
CSGetShaderResources	
CSGetUnorderedAccessViews	
CSSetConstantBuffers	
CSSetSamplers	
CSSetShader	
CSSetShaderResources	
CSSetUnorderedAccessViews	
Dispatch	
DispatchIndirect	CreateBlendState
Draw	Draw
DrawAuto	DrawAuto
DrawIndexed	DrawIndexed
DrawIndexedInstanced	DrawIndexedInstanced
DrawIndexedInstancedIndirect	
DrawInstanced	DrawInstanced
DrawInstancedIndirect	
DSGetConstantBuffers	
DSGetSamplers	
DSGetShader	
DSGetShaderResources	

DSSetConstantBuffers	
DSSetSamplers	
DSSetShader	
DSSetShaderResources	
ExecuteCommandList	
FinishCommandList	
Flush	Flush
GenerateMips	GenerateMips
GetPredication	GetPredication
GetResourceMinLOD	
GetType	GetTextFilterSize
GSGetConstantBuffers	GSGetConstantBuffers
GSGetSamplers	GSGetSamplers
GSGetShader	GSGetShader
GSGetShaderResources	GSGetShaderResources
GSSetConstantBuffers	GSSetConstantBuffers
GSSetSamplers	GSSetSamplers
GSSetShader	GSSetShader
GSSetShaderResources	GSSetShaderResources
HSGetConstantBuffers	
HSGetSamplers	
HSGetShader	
HSGetShaderResources	
HSSetConstantBuffers	
HSSetSamplers	
HSSetShader	

HSSetShaderResources	
IAGetIndexBuffer	IAGetIndexBuffer
IAGetInputLayout	IAGetInputLayout
IAGetPrimitiveTopology	IAGetPrimitiveTopology
IAGetVertexBuffers	IAGetVertexBuffers
IASetIndexBuffer	IASetIndexBuffer
IASetInputLayout	IASetInputLayout
IASetPrimitiveTopology	IASetPrimitiveTopology
IASetVertexBuffers	IASetVertexBuffers
OMGetBlendState	OMGetBlendState
OMGetDepthStencilState	OMGetDepthStencilState
OMGetRenderTargets	OMGetRenderTargets
OMGetRenderTargetsAndUnorderedAccessViews	
OMSetBlendState	OMSetBlendState
OMSetDepthStencilState	OMSetDepthStencilState
OMSetRenderTargets	OMSetRenderTargets
OMSetRenderTargetsAndUnorderedAccessViews	
PSGetConstantBuffers	PSGetConstantBuffers
PSGetSamplers	PSGetSamplers
PSGetShader	PSGetShader
PSGetShaderResources	PSGetShaderResources
PSSetConstantBuffers	PSSetConstantBuffers
PSSetSamplers	PSSetSamplers
PSSetShader	PSSetShader
PSSetShaderResources	PSSetShaderResources
ResolveSubresource	ResolveSubresource
RSGetScissorRects	RSGetScissorRects
RSGetState	RSGetState

RSGetViewports	RSGetViewports
RSSetScissorRects	RSSetScissorRects
RSSetState	RSSetState
RSSetViewports	RSSetViewports
SetPredication	SetPredication
SetResourceMinLOD	
	SetTextFilterSize
SOGetTargets	SOGetTargets
SOSetTargets	SOSetTargets
UpdateSubresource	UpdateSubresource
VSGetConstantBuffers	VSGetConstantBuffers
VSGetSamplers	VSGetSamplers
VSGetShader	VSGetShader
VSGetShaderResources	VSGetShaderResources
VSSetConstantBuffers	VSSetConstantBuffers
VSSetSamplers	VSSetSamplers
VSSetShader	VSSetShader
VSSetShaderResources	VSSetShaderResources

The following table shows the immediate context methods that the runtime does not disable when the indicated context state objects are active.

Methods of <b>ID3D11DeviceContext</b> when __uuidof(ID3D10Device1) or __uuidof(ID3D10Device) is active	Methods of <b>ID3D10Device</b> when __uuidof(ID3D11Device) is active
Begin	
End	GetCreationFlags
	GetPrivateData

[GetContextFlags](#)

[GetData](#)

[Map](#)

[Unmap](#)

The following table shows the [ID3D10Device](#) interface methods that the runtime does not disable because they are not immediate context methods.

<b>Methods of <a href="#">ID3D10Device</a></b>
<a href="#">CheckCounter</a>
<a href="#">CheckCounterInfo</a>
<a href="#">Create*</a> , like <a href="#">CreateQuery</a>
<a href="#">GetDeviceRemovedReason</a>
<a href="#">GetExceptionMode</a>
<a href="#">OpenSharedResource</a>
<a href="#">SetExceptionMode</a>
<a href="#">SetPrivateData</a>
<a href="#">SetPrivateDataInterface</a>

**Windows Phone 8:** This API is supported.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h

Library

D3D11.lib

## See also

[ID3D11Device1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device1::CreateRasterizerState1 method (d3d11\_1.h)

Article 02/22/2024

Creates a rasterizer state object that informs the [rasterizer stage](#) how to behave and forces the sample count while UAV rendering or rasterizing.

## Syntax

C++

```
HRESULT CreateRasterizerState1(
    [in]           const D3D11_RASTERIZER_DESC1 *pRasterizerDesc,
    [out, optional] ID3D11RasterizerState1      **ppRasterizerState
);
```

## Parameters

[in] pRasterizerDesc

A pointer to a [D3D11\\_RASTERIZER\\_DESC1](#) structure that describes the rasterizer state.

[out, optional] ppRasterizerState

Address of a pointer to the [ID3D11RasterizerState1](#) interface for the rasterizer state object created.

## Return value

This method returns [E\\_OUTOFMEMORY](#) if there is insufficient memory to create the rasterizer state object. See [Direct3D 11 Return Codes](#) for other possible return values.

## Remarks

An app can create up to 4096 unique rasterizer state objects. For each object created, the runtime checks to see if a previous object has the same state. If such a previous object exists, the runtime will return a pointer to previous instance instead of creating a duplicate object.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11Device1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device1::GetImmediateContext1 method (d3d11\_1.h)

Article 02/22/2024

Gets an immediate context, which can play back command lists.

## Syntax

C++

```
void GetImmediateContext1(  
    [out] ID3D11DeviceContext1 **ppImmediateContext  
);
```

## Parameters

[out] ppImmediateContext

Upon completion of the method, the passed pointer to an [ID3D11DeviceContext1](#) interface pointer is initialized.

## Return value

None

## Remarks

`GetImmediateContext1` returns an [ID3D11DeviceContext1](#) object that represents an immediate context. You can use this immediate context to perform rendering that you want immediately submitted to a device. For most applications, an immediate context is the primary object that is used to draw your scene.

`GetImmediateContext1` increments the reference count of the immediate context by one. So, call [Release](#) on the returned interface pointer when you are done with it to avoid a memory leak.

## Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11Device1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device1::OpenSharedResource1 method (d3d11\_1.h)

Article 10/13/2021

Gives a device access to a shared resource that is referenced by a handle and that was created on a different device. You must have previously created the resource as shared and specified that it uses NT handles (that is, you set the [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_NTHANDLE](#) flag).

## Syntax

C++

```
HRESULT OpenSharedResource1(
    [in]    HANDLE hResource,
    [in]    REFIID returnedInterface,
    [out]   void     **ppResource
);
```

## Parameters

[in] hResource

A handle to the resource to open. For more info about this parameter, see Remarks.

[in] returnedInterface

The globally unique identifier (GUID) for the resource interface. For more info about this parameter, see Remarks.

[out] ppResource

A pointer to a variable that receives a pointer to the interface for the shared resource object to access.

## Return value

This method returns one of the [Direct3D 11 return codes](#). This method also returns E\_ACCESSDENIED if the permissions to access the resource aren't valid.

**Platform Update for Windows 7:** On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, `OpenSharedResource1` fails with `E_NOTIMPL` because `NTHANDLES` are used. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

## Remarks

The behavior of `OpenSharedResource1` is similar to the behavior of the `ID3D11Device::OpenSharedResource` method; each call to `OpenSharedResource1` to access a resource creates a new resource object. In other words, if you call `OpenSharedResource1` twice and pass the same resource handle to `hResource`, you receive two resource objects with different `IUnknown` pointers.

### To share a resource between two devices

1. Create the resource as shared and specify that it uses NT handles, by setting the `D3D11_RESOURCE_MISC_SHARED_NTHANDLE` flag.
2. Obtain the REFIID, or GUID, of the interface to the resource by using the `_uuidof()` macro. For example, `_uuidof(ID3D11Texture2D)` retrieves the GUID of the interface to a 2D texture.
3. Query the resource for the `IDXGIResource1` interface.
4. Call the `IDXGIResource1::CreateSharedHandle` method to obtain the unique handle to the resource.

## Examples

### syntax

```
HANDLE handle = GetSharedHandleFromOtherProcess();
ID3D11Device1* pDevice;
ID3D11Texture2D* pTexture2D;

pDevice-&gt;OpenSharedResource1(
    handle,
    _uuidof(ID3D11Texture2D),
    (void**)&pTexture2D);
```

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11Device1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device1::OpenSharedResourceByName method (d3d11\_1.h)

Article 02/22/2024

Gives a device access to a shared resource that is referenced by name and that was created on a different device. You must have previously created the resource as shared and specified that it uses NT handles (that is, you set the [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_NTHANDLE](#) flag).

## Syntax

C++

```
HRESULT OpenSharedResourceByName(
    [in]  LPCWSTR lpName,
    [in]  DWORD    dwDesiredAccess,
    [in]  REFIID   returnedInterface,
    [out] void     **ppResource
);
```

## Parameters

[in] lpName

The name of the resource to open. This parameter cannot be **NULL**.

[in] dwDesiredAccess

The requested access rights to the resource. In addition to the [generic access rights](#), DXGI defines the following values:

- **DXGI\_SHARED\_RESOURCE\_READ** ( 0x80000000L ) - specifies read access to the resource.
- **DXGI\_SHARED\_RESOURCE\_WRITE** ( 1 ) - specifies write access to the resource.

You can combine values by using a bitwise **OR** operation.

[in] returnedInterface

The globally unique identifier (GUID) for the resource interface. For more info, see Remarks.

[out] ppResource

A pointer to a variable that receives a pointer to the interface for the shared resource object to access.

## Return value

This method returns one of the [Direct3D 11 return codes](#). This method also returns E\_ACCESSDENIED if the permissions to access the resource aren't valid.

**Platform Update for Windows 7:** On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **OpenSharedResourceByName** fails with E\_NOTIMPL because NTHANDLES are used. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

## Remarks

The behavior of **OpenSharedResourceByName** is similar to the behavior of the [ID3D11Device1::OpenSharedResource1](#) method; each call to **OpenSharedResourceByName** to access a resource creates a new resource object. In other words, if you call **OpenSharedResourceByName** twice and pass the same resource name to *lpName*, you receive two resource objects with different [IUnknown](#) pointers.

### To share a resource between two devices

1. Create the resource as shared and specify that it uses NT handles, by setting the [D3D11\\_RESOURCE\\_MISC\\_SHARED\\_NTHANDLE](#) flag.
2. Obtain the REFIID, or GUID, of the interface to the resource by using the [\\_uuidof\(\)](#) macro. For example, [\\_uuidof\(ID3D11Texture2D\)](#) retrieves the GUID of the interface to a 2D texture.
3. Query the resource for the [IDXGIResource1](#) interface.
4. Call the [IDXGIResource1::CreateSharedHandle](#) method to obtain the unique handle to the resource. In this [IDXGIResource1::CreateSharedHandle](#) call, you must pass a name for the resource if you want to subsequently call **OpenSharedResourceByName** to access the resource by name.

## Examples

### Syntax

```
ID3D11Device1* pDevice;  
ID3D11Texture2D* pTexture2D;
```

```
pDevice->OpenSharedResourceByName(  
    "MySurface",  
    DXGI_SHARED_RESOURCE_READ,  
    __uuidof(ID3D11Texture2D),  
    (void**)&pTexture2D);
```

# Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11Device1](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device2 interface (d3d11\_2.h)

Article 07/27/2022

The device interface represents a virtual adapter; it is used to create resources. ID3D11Device2 adds new methods to those in [ID3D11Device1](#).

## Inheritance

The ID3D11Device2 interface inherits from [ID3D11Device1](#). ID3D11Device2 also has these types of members:

## Methods

The ID3D11Device2 interface has these methods.

<a href="#">ID3D11Device2::CheckMultisampleQualityLevels1</a>
Get the number of quality levels available during multisampling. (ID3D11Device2.CheckMultisampleQualityLevels1)
<a href="#">ID3D11Device2::CreateDeferredContext2</a>
Creates a deferred context, which can record command lists. (ID3D11Device2.CreateDeferredContext2)
<a href="#">ID3D11Device2::GetImmediateContext2</a>
Gets an immediate context, which can play back command lists. (ID3D11Device2.GetImmediateContext2)
<a href="#">ID3D11Device2::GetResourceTiling</a>
Gets info about how a tiled resource is broken into tiles. (ID3D11Device2.GetResourceTiling)

## Requirements

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]

Target Platform	Windows
Header	d3d11_2.h

## See also

[Core Interfaces](#)

[ID3D11Device1](#)

---

## Feedback

Was this page helpful?



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

# ID3D11Device2::CheckMultisampleQualityLevels1 method (d3d11\_2.h)

Article 02/22/2024

Get the number of quality levels available during multisampling.

## Syntax

C++

```
HRESULT CheckMultisampleQualityLevels1(
    [in] DXGI_FORMAT Format,
    [in] UINT SampleCount,
    [in] UINT Flags,
    [out] UINT *pNumQualityLevels
);
```

## Parameters

[in] Format

Type: [DXGI\\_FORMAT](#)

The texture format during multisampling.

[in] SampleCount

Type: [UINT](#)

The number of samples during multisampling.

[in] Flags

Type: [UINT](#)

A combination of [D3D11\\_CHECK\\_MULTISAMPLE\\_QUALITY\\_LEVELS\\_FLAGS](#) values that are combined by using a bitwise OR operation. Currently, only [D3D11\\_CHECK\\_MULTISAMPLE\\_QUALITY\\_LEVELS\\_TILED\\_RESOURCE](#) is supported.

[out] pNumQualityLevels

Type: [UINT\\*](#)

A pointer to a variable the receives the number of quality levels supported by the adapter. See Remarks.

## Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

When you multisample a texture, the number of quality levels available for an adapter is dependent on the texture format that you use and the number of samples that you request. The maximum number of quality levels is defined by [D3D11\\_MAX\\_MULTISAMPLE\\_SAMPLE\\_COUNT](#) in D3D11.h. If this method returns 0, the format and sample count combination is not supported for the installed adapter.

Furthermore, the definition of a quality level is up to each hardware vendor to define, however no facility is provided by Direct3D to help discover this information.

Note that FEATURE\_LEVEL\_10\_1 devices are required to support 4x MSAA for all render targets except R32G32B32A32 and R32G32B32. FEATURE\_LEVEL\_11\_0 devices are required to support 4x MSAA for all render target formats, and 8x MSAA for all render target formats except R32G32B32A32 formats.

## Requirements

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

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device2::CreateDeferredContext2 method (d3d11\_2.h)

Article 02/22/2024

Creates a deferred context, which can record [command lists](#).

## Syntax

C++

```
HRESULT CreateDeferredContext2(
    UINT ContextFlags,
    [out, optional] ID3D11DeviceContext2 **ppDeferredContext
);
```

## Parameters

ContextFlags

Type: [UINT](#)

Reserved for future use. Pass 0.

[out, optional] ppDeferredContext

Type: [ID3D11DeviceContext2\\*\\*](#)

Upon completion of the method, the passed pointer to an [ID3D11DeviceContext2](#) interface pointer is initialized.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the following:

- Returns [DXGI\\_ERROR\\_DEVICE\\_REMOVED](#) if the video card has been physically removed from the system, or a driver upgrade for the video card has occurred. If this error occurs, you should destroy and recreate the device.
- Returns [DXGI\\_ERROR\\_INVALID\\_CALL](#) if the [CreateDeferredContext2](#) method can't be called from the current context. For example, if the device was created with the

`D3D11_CREATE_DEVICE_SINGLETHREADED` value, `CreateDeferredContext2` returns `DXGI_ERROR_INVALID_CALL`.

- Returns `E_INVALIDARG` if the `ContextFlags` parameter is invalid.
- Returns `E_OUTOFMEMORY` if the app has exhausted available memory.

## Remarks

A deferred context is a thread-safe context that you can use to record graphics commands on a thread other than the main rendering thread. By using a deferred context, you can record graphics commands into a command list that is encapsulated by the `ID3D11CommandList` interface. After you record all scene items, you can then submit them to the main render thread for final rendering. In this manner, you can perform rendering tasks concurrently across multiple threads and potentially improve performance in multi-core CPU scenarios.

You can create multiple deferred contexts.

**Note** If you use the `D3D11_CREATE_DEVICE_SINGLETHREADED` value to create the device, `CreateDeferredContext2` fails with `DXGI_ERROR_INVALID_CALL`, and you can't create a deferred context.

For more information about deferred contexts, see [Immediate and Deferred Rendering](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11Device1::CreateDeferredContext1](#)

[ID3D11Device2](#)

[ID3D11Device3::CreateDeferredContext3](#)

[ID3D11Device::CreateDeferredContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device2::GetImmediateContext2 method (d3d11\_2.h)

Article02/22/2024

Gets an immediate context, which can play back [command lists](#).

## Syntax

C++

```
void GetImmediateContext2(
    [out] ID3D11DeviceContext2 **ppImmediateContext
);
```

## Parameters

[out] ppImmediateContext

Type: [ID3D11DeviceContext2\\*\\*](#)

Upon completion of the method, the passed pointer to an [ID3D11DeviceContext2](#) interface pointer is initialized.

## Return value

None

## Remarks

The **GetImmediateContext2** method returns an [ID3D11DeviceContext2](#) object that represents an immediate context, which is used to perform rendering that you want immediately submitted to a device. For most apps, an immediate context is the primary object that is used to draw your scene.

The **GetImmediateContext2** method increments the reference count of the immediate context by one. Therefore, you must call [Release](#) on the returned interface pointer when you are done with it to avoid a memory leak.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11Device2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device2::GetResourceTiling method (d3d11\_2.h)

Article 02/22/2024

Gets info about how a tiled resource is broken into tiles.

## Syntax

C++

```
void GetResourceTiling(
    [in]                  ID3D11Resource           *pTiledResource,
    [out, optional]        UINT                 *pNumTilesForEntireResource,
    [out, optional]        D3D11_PACKED_MIP_DESC *pPackedMipDesc,
    [out, optional]        D3D11_TILE_SHAPE
    *pStandardTileShapeForNonPackedMips,
    [in, out, optional]   UINT                 *pNumSubresourceTilings,
    [in]                  UINT                 FirstSubresourceTilingToGet,
    [out]                 D3D11_SUBRESOURCE_TILING
    *pSubresourceTilingsForNonPackedMips
);
```

## Parameters

[in] pTiledResource

Type: [ID3D11Resource\\*](#)

A pointer to the tiled resource to get info about.

[out, optional] pNumTilesForEntireResource

Type: [UINT\\*](#)

A pointer to a variable that receives the number of tiles needed to store the entire tiled resource.

[out, optional] pPackedMipDesc

Type: [D3D11\\_PACKED\\_MIP\\_DESC\\*](#)

A pointer to a [D3D11\\_PACKED\\_MIP\\_DESC](#) structure that **GetResourceTiling** fills with info about how the tiled resource's mipmaps are packed.

[out, optional] *pStandardTileShapeForNonPackedMips*

Type: [D3D11\\_TILE\\_SHAPE\\*](#)

A pointer to a [D3D11\\_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 [D3D11\\_TILE\\_SHAPE](#) to zeros.

[in, out, optional] *pNumSubresourceTilings*

Type: [UINT\\*](#)

A pointer to a variable that contains the number of tiles in the subresource. On input, this is the number of subresources to query tilings for; on output, this is the number that was actually retrieved at *pSubresourceTilingsForNonPackedMips* (clamped to what's available).

[in] *FirstSubresourceTilingToGet*

Type: [UINT](#)

The number of the first subresource tile to get. [GetResourceTiling](#) ignores this parameter if the number that *pNumSubresourceTilings* points to is 0.

[out] *pSubresourceTilingsForNonPackedMips*

Type: [D3D11\\_SUBRESOURCE\\_TILING\\*](#)

A pointer to a [D3D11\\_SUBRESOURCE\\_TILING](#) structure that [GetResourceTiling](#) fills with info about subresource tiles.

If subresource tiles are part of packed mipmaps, [GetResourceTiling](#) sets the members of [D3D11\\_SUBRESOURCE\\_TILING](#) to zeros, except the [StartTileIndexInOverallResource](#) member, which [GetResourceTiling](#) sets to [D3D11\\_PACKED\\_TILE](#) (0xffffffff). The [D3D11\\_PACKED\\_TILE](#) constant indicates that the whole [D3D11\\_SUBRESOURCE\\_TILING](#) structure is meaningless for this situation, and the info that the *pPackedMipDesc* parameter points to applies.

## Return value

None

# Remarks

For more info about tiled resources, see [Tiled resources](#).

# Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11Device2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3 interface (d3d11\_3.h)

Article 02/22/2024

The device interface represents a virtual adapter; it is used to create resources. ID3D11Device3 adds new methods to those in [ID3D11Device2](#).

## Inheritance

The ID3D11Device3 interface inherits from [ID3D11Device2](#). ID3D11Device3 also has these types of members:

## Methods

The ID3D11Device3 interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11Device3::CreateDeferredContext3</a>
Creates a deferred context, which can record command lists. (ID3D11Device3.CreateDeferredContext3)
<a href="#">ID3D11Device3::CreateQuery1</a>
Creates a query object for querying information from the graphics processing unit (GPU).
<a href="#">ID3D11Device3::CreateRasterizerState2</a>
Creates a rasterizer state object that informs the rasterizer stage how to behave and forces the sample count while UAV rendering or rasterizing. (ID3D11Device3.CreateRasterizerState2)
<a href="#">ID3D11Device3::CreateRenderTargetView1</a>
Creates a render-target view for accessing resource data. (ID3D11Device3.CreateRenderTargetView1)
<a href="#">ID3D11Device3::CreateShaderResourceView1</a>
Creates a shader-resource view for accessing data in a resource. (ID3D11Device3.CreateShaderResourceView1)
<a href="#">ID3D11Device3::CreateTexture2D1</a>

Creates a 2D texture.
<a href="#">ID3D11Device3::CreateTexture3D1</a>
Creates a 3D texture.
<a href="#">ID3D11Device3::CreateUnorderedAccessView1</a>
Creates a view for accessing an unordered access resource. (ID3D11Device3.CreateUnorderedAccessView1)
<a href="#">ID3D11Device3::GetImmediateContext3</a>
Gets an immediate context, which can play back command lists. (ID3D11Device3.GetImmediateContext3)
<a href="#">ID3D11Device3::ReadFromSubresource</a>
Copies data from a D3D11_USAGE_DEFAULTtexture which was mapped using ID3D11DeviceContext3::Mapwhile providing a NULL D3D11_MAPPED_SUBRESOURCEparameter.
<a href="#">ID3D11Device3::WriteToSubresource</a>
Copies data into a D3D11_USAGE_DEFAULTtexture which was mapped using ID3D11DeviceContext3::Mapwhile providing a NULL D3D11_MAPPED_SUBRESOURCEparameter.

## Requirements

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

Requirement	Value
Minimum supported client	Windows 10 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_3.h

## See also

[Core Interfaces](#)

[ID3D11Device2](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::CreateDeferredContext3 method (d3d11\_3.h)

Article 02/22/2024

Creates a deferred context, which can record [command lists](#).

## Syntax

C++

```
HRESULT CreateDeferredContext3(
    UINT ContextFlags,
    [out, optional] ID3D11DeviceContext3 **ppDeferredContext
);
```

## Parameters

ContextFlags

Type: **UINT**

Reserved for future use. Pass 0.

[out, optional] ppDeferredContext

Type: **ID3D11DeviceContext3\*\***

Upon completion of the method, the passed pointer to an [ID3D11DeviceContext3](#) interface pointer is initialized.

## Return value

Type: [\*\*HRESULT\*\*](#)

Returns S\_OK if successful; otherwise, returns one of the following:

- Returns **DXGI\_ERROR\_DEVICE\_REMOVED** if the video card has been physically removed from the system, or a driver upgrade for the video card has occurred. If this error occurs, you should destroy and recreate the device.
- Returns **DXGI\_ERROR\_INVALID\_CALL** if the **CreateDeferredContext3** method can't be called from the current context. For example, if the device was created with the

`D3D11_CREATE_DEVICE_SINGLETHREADED` value, `CreateDeferredContext3` returns `DXGI_ERROR_INVALID_CALL`.

- Returns `E_INVALIDARG` if the `ContextFlags` parameter is invalid.
- Returns `E_OUTOFMEMORY` if the app has exhausted available memory.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device1::CreateDeferredContext1](#)

[ID3D11Device2::CreateDeferredContext2](#)

[ID3D11Device3](#)

[ID3D11Device::CreateDeferredContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::CreateQuery1 method (d3d11\_3.h)

Article 02/22/2024

Creates a query object for querying information from the graphics processing unit (GPU).

## Syntax

C++

```
HRESULT CreateQuery1(
    [in]           const D3D11_QUERY_DESC1 *pQueryDesc1,
    [out, optional] ID3D11Query1          **ppQuery1
);
```

## Parameters

[in] pQueryDesc1

Type: [const D3D11\\_QUERY\\_DESC1\\*](#)

Pointer to a [D3D11\\_QUERY\\_DESC1](#) structure that represents a query description.

[out, optional] ppQuery1

Type: [ID3D11Query1\\*\\*](#)

A pointer to a memory block that receives a pointer to a [ID3D11Query1](#) interface for the created query object. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

## Return value

Type: [HRESULT](#)

This method returns **E\_OUTOFMEMORY** if there is insufficient memory to create the query object.

See [Direct3D 11 Return Codes](#) for other possible return values.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::CreateRasterizerState2 method (d3d11\_3.h)

Article 07/27/2022

Creates a rasterizer state object that informs the [rasterizer stage](#) how to behave and forces the sample count while UAV rendering or rasterizing.

## Syntax

C++

```
HRESULT CreateRasterizerState2(
    [in]           const D3D11_RASTERIZER_DESC2 *pRasterizerDesc,
    [out, optional] ID3D11RasterizerState2      **ppRasterizerState
);
```

## Parameters

[in] pRasterizerDesc

Type: [const D3D11\\_RASTERIZER\\_DESC2\\*](#)

A pointer to a [D3D11\\_RASTERIZER\\_DESC2](#) structure that describes the rasterizer state.

[out, optional] ppRasterizerState

Type: [ID3D11RasterizerState2\\*\\*](#)

A pointer to a memory block that receives a pointer to a [ID3D11RasterizerState2](#) interface for the created rasterizer state object. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

## Return value

Type: [HRESULT](#)

This method returns **E\_OUTOFMEMORY** if there is insufficient memory to create the rasterizer state object. See [Direct3D 11 Return Codes](#) for other possible return values.

# Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::CreateRenderTargetView

## 1 method (d3d11\_3.h)

Article 07/27/2022

Creates a render-target view for accessing resource data.

## Syntax

C++

```
HRESULT CreateRenderTargetView1(
    [in]           ID3D11Resource                  *pResource,
    [in, optional] const D3D11_RENDER_TARGET_VIEW_DESC1 *pDesc1,
    [out, optional] ID3D11RenderTargetView1          **ppRTView1
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

Pointer to a [ID3D11Resource](#) that represents a render target. This resource must have been created with the [D3D11\\_BIND\\_RENDER\\_TARGET](#) flag.

[in, optional] pDesc1

Type: [const D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC1\\*](#)

Pointer to a [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC1](#) that represents a render-target view description. Set this parameter to **NULL** to create a view that accesses all of the subresources in mipmap level 0.

[out, optional] ppRTView1

Type: [ID3D11RenderTargetView1\\*\\*](#)

A pointer to a memory block that receives a pointer to a [ID3D11RenderTargetView1](#) interface for the created render-target view. Set this parameter to **NULL** to validate the other input parameters (the method will return [S\\_FALSE](#) if the other input parameters pass validation).

# Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

A render-target view can be bound to the output-merger stage by calling [ID3D11DeviceContext::OMSetRenderTargets](#).

## Requirements

<b>Minimum supported client</b>	Windows 10 [desktop apps only]
<b>Minimum supported server</b>	Windows Server 2016 [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	d3d11_3.h
<b>Library</b>	D3D11.lib

## See also

[ID3D11Device3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::CreateShaderResourceView1 method (d3d11\_3.h)

Article 02/22/2024

Creates a shader-resource view for accessing data in a resource.

## Syntax

C++

```
HRESULT CreateShaderResourceView1(
    [in]          ID3D11Resource                  *pResource,
    [in, optional] const D3D11_SHADER_RESOURCE_VIEW_DESC1 *pDesc1,
    [out, optional] ID3D11ShaderResourceView1           **ppSRView1
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

Pointer to the resource that will serve as input to a shader. This resource must have been created with the [D3D11\\_BIND\\_SHADER\\_RESOURCE](#) flag.

[in, optional] pDesc1

Type: [const D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC1\\*](#)

A pointer to a [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC1](#) structure that describes a shader-resource view. Set this parameter to **NULL** to create a view that accesses the entire resource (using the format the resource was created with).

[out, optional] ppSRView1

Type: [ID3D11ShaderResourceView1\\*\\*](#)

A pointer to a memory block that receives a pointer to a [ID3D11ShaderResourceView1](#) interface for the created shader-resource view. Set this parameter to **NULL** to validate the other input parameters (the method will return [S\\_FALSE](#) if the other input parameters pass validation).

# Return value

Type: [HRESULT](#)

This method returns `E_OUTOFMEMORY` if there is insufficient memory to create the shader-resource view. See [Direct3D 11 Return Codes](#) for other possible return values.

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device3](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::CreateTexture2D1 method (d3d11\_3.h)

Article 02/12/2024

Creates a [2D texture](#).

## Syntax

C++

```
HRESULT CreateTexture2D1(
    [in]          const D3D11_TEXTURE2D_DESC1  *pDesc1,
    [in, optional] const D3D11_SUBRESOURCE_DATA *pInitialData,
    [out, optional] ID3D11Texture2D1           **ppTexture2D
);
```

## Parameters

[in] `pDesc1`

Type: [const D3D11\\_TEXTURE2D\\_DESC1\\*](#)

A pointer to a [D3D11\\_TEXTURE2D\\_DESC1](#) structure that describes a 2D texture resource. To create a typeless resource that can be interpreted at runtime into different, compatible formats, specify a typeless format in the texture description. To generate mipmap levels automatically, set the number of mipmap levels to 0.

[in, optional] `pInitialData`

Type: [const D3D11\\_SUBRESOURCE\\_DATA\\*](#)

A pointer to an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures that describe subresources for the 2D texture resource. Applications can't specify **NULL** for *pInitialData* when creating IMMUTABLE resources (see [D3D11\\_USAGE](#)). If the resource is multisampled, *pInitialData* must be **NULL** because multisampled resources can't be initialized with data when they're created.

If you don't pass anything to *pInitialData*, the initial content of the memory for the resource is undefined. In this case, you need to write the resource content some other way before the resource is read.

You can determine the size of this array from values in the **MipLevels** and **ArraySize** members of the **D3D11\_TEXTURE2D\_DESC1** structure to which *pDesc1* points by using the following calculation:

$$\text{MipLevels} * \text{ArraySize}$$

For more info about this array size, see Remarks.

[out, optional] *ppTexture2D*

Type: [ID3D11Texture2D1\\*\\*](#)

A pointer to a memory block that receives a pointer to a [ID3D11Texture2D1](#) interface for the created texture. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

## Return value

Type: [HRESULT](#)

If the method succeeds, the return code is **S\_OK**. See [Direct3D 11 Return Codes](#) for failing error codes.

## Remarks

**CreateTexture2D1** creates a 2D texture resource, which can contain a number of 2D subresources. The number of subresources is specified in the texture description. All textures in a resource must have the same format, size, and number of mipmap levels.

All resources are made up of one or more subresources. To load data into the texture, applications can supply the data initially as an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures pointed to by *pInitialData*, or they can use one of the D3DX texture functions such as [D3DX11CreateTextureFromFile](#).

For a 32 x 32 texture with a full mipmap chain, the *pInitialData* array has the following 6 elements:

- *pInitialData[0]* = 32x32
- *pInitialData[1]* = 16x16
- *pInitialData[2]* = 8x8
- *pInitialData[3]* = 4x4
- *pInitialData[4]* = 2x2
- *pInitialData[5]* = 1x1

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::CreateTexture3D1 method (d3d11\_3.h)

Article 10/13/2021

Creates a [3D texture](#).

## Syntax

C++

```
HRESULT CreateTexture3D1(
    [in]          const D3D11_TEXTURE3D_DESC1  *pDesc1,
    [in, optional] const D3D11_SUBRESOURCE_DATA *pInitialData,
    [out, optional] ID3D11Texture3D1           **ppTexture3D
);
```

## Parameters

[in] pDesc1

Type: [const D3D11\\_TEXTURE3D\\_DESC1\\*](#)

A pointer to a [D3D11\\_TEXTURE3D\\_DESC1](#) structure that describes a 3D texture resource. To create a typeless resource that can be interpreted at runtime into different, compatible formats, specify a typeless format in the texture description. To generate mipmap levels automatically, set the number of mipmap levels to 0.

[in, optional] pInitialData

Type: [const D3D11\\_SUBRESOURCE\\_DATA\\*](#)

A pointer to an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures that describe subresources for the 3D texture resource. Applications can't specify **NULL** for *pInitialData* when creating IMMUTABLE resources (see [D3D11\\_USAGE](#)). If the resource is multisampled, *pInitialData* must be **NULL** because multisampled resources can't be initialized with data when they are created.

If you don't pass anything to *pInitialData*, the initial content of the memory for the resource is undefined. In this case, you need to write the resource content some other way before the resource is read.

You can determine the size of this array from the value in the **MipLevels** member of the **D3D11\_TEXTURE3D\_DESC1** structure to which *pDesc1* points. Arrays of 3D volume textures aren't supported.

For more information about this array size, see Remarks.

[out, optional] *ppTexture3D*

Type: [ID3D11Texture3D1\\*\\*](#)

A pointer to a memory block that receives a pointer to a [ID3D11Texture3D1](#) interface for the created texture. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

## Return value

Type: [HRESULT](#)

If the method succeeds, the return code is **S\_OK**. See [Direct3D 11 Return Codes](#) for failing error codes.

## Remarks

`CreateTexture3D1` creates a 3D texture resource, which can contain a number of 3D subresources. The number of textures is specified in the texture description. All textures in a resource must have the same format, size, and number of mipmap levels.

All resources are made up of one or more subresources. To load data into the texture, applications can supply the data initially as an array of [D3D11\\_SUBRESOURCE\\_DATA](#) structures pointed to by *pInitialData*, or they can use one of the D3DX texture functions such as [D3DX11CreateTextureFromFile](#).

Each element of *pInitialData* provides all of the slices that are defined for a given mipmap level. For example, for a 32 x 32 x 4 volume texture with a full mipmap chain, the array has the following 6 elements:

- *pInitialData[0]* = 32x32 with 4 slices
- *pInitialData[1]* = 16x16 with 2 slices
- *pInitialData[2]* = 8x8 with 1 slice
- *pInitialData[3]* = 4x4 with 1 slice
- *pInitialData[4]* = 2x2 with 1 slice
- *pInitialData[5]* = 1x1 with 1 slice

# Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::CreateUnorderedAccessView1 method (d3d11\_3.h)

Article 02/22/2024

Creates a view for accessing an [unordered access](#) resource.

## Syntax

C++

```
HRESULT CreateUnorderedAccessView1(
    [in]          ID3D11Resource                  *pResource,
    [in, optional] const D3D11_UNORDERED_ACCESS_VIEW_DESC1 *pDesc1,
    [out, optional] ID3D11UnorderedAccessView1           **ppUAVview1
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

Pointer to an [ID3D11Resource](#) that represents a resources that will serve as an input to a shader.

[in, optional] pDesc1

Type: [const D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC1\\*](#)

Pointer to a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC1](#) structure that represents an unordered-access view description. Set this parameter to **NULL** to create a view that accesses the entire resource (using the format the resource was created with).

[out, optional] ppUAVview1

Type: [ID3D11UnorderedAccessView1\\*\\*](#)

A pointer to a memory block that receives a pointer to a [ID3D11UnorderedAccessView1](#) interface for the created unordered-access view. Set this parameter to **NULL** to validate the other input parameters (the method will return **S\_FALSE** if the other input parameters pass validation).

# Return value

Type: [HRESULT](#)

This method returns `E_OUTOFMEMORY` if there is insufficient memory to create the unordered-access view. See [Direct3D 11 Return Codes](#) for other possible return values.

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device3](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::GetImmediateContext3 method (d3d11\_3.h)

Article 02/22/2024

Gets an immediate context, which can play back [command lists](#).

## Syntax

C++

```
void GetImmediateContext3(
    [out] ID3D11DeviceContext3 **ppImmediateContext
);
```

## Parameters

[out] ppImmediateContext

Type: [ID3D11DeviceContext3\\*\\*](#)

Upon completion of the method, the passed pointer to an [ID3D11DeviceContext3](#) interface pointer is initialized.

## Return value

None

## Remarks

The **GetImmediateContext3** method outputs an [ID3D11DeviceContext3](#) object that represents an immediate context, which is used to perform rendering that you want immediately submitted to a device. For most apps, an immediate context is the primary object that is used to draw your scene.

The **GetImmediateContext3** method increments the reference count of the immediate context by one. Therefore, you must call [Release](#) on the returned interface pointer when you are done with it to avoid a memory leak.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device1::GetImmediateContext1](#)

[ID3D11Device2::GetImmediateContext2](#)

[ID3D11Device3](#)

[ID3D11Device::GetImmediateContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::ReadFromSubresource method (d3d11\_3.h)

Article 10/13/2021

Copies data from a [D3D11\\_USAGE\\_DEFAULT](#) texture which was mapped using [ID3D11DeviceContext3::Map](#) while providing a NULL [D3D11\\_MAPPED\\_SUBRESOURCE](#) parameter.

## Syntax

C++

```
void ReadFromSubresource(
    [out]          void      *pDstData,
    [in]           UINT       DstRowPitch,
    [in]           UINT       DstDepthPitch,
    [in]           ID3D11Resource *pSrcResource,
    [in]           UINT       SrcSubresource,
    [in, optional] const D3D11_BOX *pSrcBox
);
```

## Parameters

[out] pDstData

Type: [void\\*](#)

A pointer to the destination data in memory.

[in] DstRowPitch

Type: [UINT](#)

The size of one row of the destination data.

[in] DstDepthPitch

Type: [UINT](#)

The size of one depth slice of destination data.

[in] pSrcResource

Type: [ID3D11Resource](#)\*

A pointer to the source resource (see [ID3D11Resource](#)).

[in] SrcSubresource

Type: [UINT](#)

A zero-based index, that identifies the destination subresource. For more details, see [D3D11CalcSubresource](#).

[in, optional] pSrcBox

Type: [const D3D11\\_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 [D3D11\\_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

None

## Remarks

The provided resource must be a [D3D11\\_USAGE\\_DEFAULT](#) texture which was mapped for writing by a previous call to [ID3D11DeviceContext3::Map](#) while providing a NULL [D3D11\\_MAPPED\\_SUBRESOURCE](#) parameter.

This API is intended for calling at high frequency. Callers can reduce memory by making iterative calls that update progressive regions of the texture, while provide a small buffer during each call. It is most efficient to specify large enough regions, though, because this enables D3D to fill whole cache lines in the texture before returning.

For efficiency, ensure the bounds and alignment of the extents within the box are ( 64 / [Bytes per pixel] ) pixels horizontally. Vertical bounds and alignment should be 2 rows, except when 1-byte-per-pixel formats are used, in which case 4 rows are recommended.

Single depth slices per call are handled efficiently. It is recommended but not necessary to provide pointers and strides which are 128-byte aligned.

When reading from 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.

## Requirements

Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device3::WriteToSubresource method (d3d11\_3.h)

Article 10/13/2021

Copies data into a [D3D11\\_USAGE\\_DEFAULT](#) texture which was mapped using [ID3D11DeviceContext3::Map](#) while providing a NULL [D3D11\\_MAPPED\\_SUBRESOURCE](#) parameter.

## Syntax

C++

```
void WriteToSubresource(
    [in]          ID3D11Resource  *pDstResource,
    [in]          UINT           DstSubresource,
    [in, optional] const D3D11_BOX *pDstBox,
    [in]          const void     *pSrcData,
    [in]          UINT           SrcRowPitch,
    [in]          UINT           SrcDepthPitch
);
```

## Parameters

[in] `pDstResource`

Type: [ID3D11Resource\\*](#)

A pointer to the destination resource (an [ID3D11Resource](#)).

[in] `DstSubresource`

Type: [UINT](#)

A zero-based index, that identifies the destination subresource. For more details, see [D3D11CalcSubresource](#).

[in, optional] `pDstBox`

Type: [const D3D11\\_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 [D3D11\\_BOX](#)).

An empty box results in a no-op. A box is empty if the top value is greater than or equal to the bottom value, or the left value is greater than or equal to the right value, or the front value is greater than or equal to the back value. When the box is empty, this method doesn't perform any operation.

[in] pSrcData

Type: **const void\***

A pointer to the source data in memory.

[in] SrcRowPitch

Type: **UINT**

The size of one row of the source data.

[in] SrcDepthPitch

Type: **UINT**

The size of one depth slice of source data.

## Return value

None

## Remarks

The provided resource must be a [D3D11\\_USAGE\\_DEFAULT](#) texture which was mapped for writing by a previous call to [ID3D11DeviceContext3::Map](#) while providing a NULL [D3D11\\_MAPPED\\_SUBRESOURCE](#) parameter.

This API is intended for calling at high frequency. Callers can reduce memory by making iterative calls that update progressive regions of the texture, while provide a small buffer during each call. It is most efficient to specify large enough regions, though, because this enables D3D to fill whole cache lines in the texture before returning.

For efficiency, ensure the bounds and alignment of the extents within the box are ( 64 / [bytes per pixel] ) pixels horizontally. Vertical bounds and alignment should be 2 rows, except when 1-byte-per-pixel formats are used, in which case 4 rows are recommended.

Single depth slices per call are handled efficiently. It is recommended but not necessary to provide pointers and strides which are 128-byte aligned.

When writing to sub mipmap levels, it is recommended to use larger width and heights than described above. This is because small mipmap levels may actually be stored within a larger block of memory, with an opaque amount of offsetting which can interfere with alignment to cache lines.

## Requirements

Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11Device3](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device4 interface (d3d11\_4.h)

Article 07/22/2021

The device interface represents a virtual adapter; it is used to create resources. **ID3D11Device4** adds new methods to those in [ID3D11Device3](#), such as **RegisterDeviceRemovedEvent** and **UnregisterDeviceRemoved**.

## Inheritance

The **ID3D11Device4** interface inherits from [ID3D11Device3](#). **ID3D11Device4** also has these types of members:

## Methods

The **ID3D11Device4** interface has these methods.

<a href="#">ID3D11Device4::RegisterDeviceRemovedEvent</a>
Registers the "device removed" event and indicates when a Direct3D device has become removed for any reason, using an asynchronous notification mechanism.

<a href="#">ID3D11Device4::UnregisterDeviceRemoved</a>
Unregisters the "device removed" event.

## Requirements

Target Platform	Windows
Header	d3d11_4.h

## See also

[Core Interfaces](#)

[ID3D11Device3](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device4::RegisterDeviceRemoved Event method (d3d11\_4.h)

Article02/22/2024

Registers the "device removed" event and indicates when a Direct3D device has become removed for any reason, using an asynchronous notification mechanism.

## Syntax

C++

```
HRESULT RegisterDeviceRemovedEvent(
    [in]  HANDLE hEvent,
    [out] DWORD  *pdwCookie
);
```

## Parameters

[in] hEvent

Type: **HANDLE**

The handle to the "device removed" event.

[out] pdwCookie

Type: **DWORD\***

A pointer to information about the "device removed" event, which can be used in [UnregisterDeviceRemoved](#) to unregister the event.

## Return value

Type: **HRESULT**

See [Direct3D 11 Return Codes](#).

## Remarks

Indicates when a Direct3D device has become removed for any reason, using an asynchronous notification mechanism, rather than as an HRESULT from [Present](#). The reason for device removal can be retrieved using [ID3D11Device::GetDeviceRemovedReason](#) after being notified of the occurrence.

Applications register and un-register a Win32 event handle with a particular device. That event handle will be signaled when the device becomes removed. A poll into the device's [ID3D11Device::GetDeviceRemovedReason](#) method indicates that the device is removed.

[ISignalableNotifier](#) or [SetThreadpoolWait](#) can be used by UWP apps.

When the graphics device is lost, the app or title will receive the graphics event, so that the app or title knows that its graphics device is no longer valid and it is safe for the app or title to re-create its DirectX devices. In response to this event, the app or title needs to re-create its rendering device and pass it into a SetRenderingDevice call on the composition graphics device objects.

After setting this new rendering device, the app or title needs to redraw content of all the pre-existing surfaces after the composition graphics device's [OnRenderingDeviceReplaced](#) event is fired.

This method supports Composition for device loss.

The event is not signaled when it is most ideal to re-create. So, instead, we recommend iterating through the adapter ordinals and creating the first ordinal that will succeed.

The application can register an event with the device. The application will be signaled when the device becomes removed.

If the device is already removed, calls to [RegisterDeviceRemovedEvent](#) will signal the event immediately. No device-removed error code will be returned from [RegisterDeviceRemovedEvent](#).

Each "device removed" event is never signaled, or is signaled only once. These events are not signaled during device destruction. These events are unregistered during destruction.

The semantics of [RegisterDeviceRemovedEvent](#) are similar to [IDXGIFactory2::RegisterOcclusionStatusEvent](#).

## Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_4.h
Library	D3d11.lib

## See also

[ID3D11Device4](#)

[UnregisterDeviceRemoved](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device4::UnregisterDeviceRemoved method (d3d11\_4.h)

Article 02/22/2024

Unregisters the "device removed" event.

## Syntax

C++

```
void UnregisterDeviceRemoved(
    [in] DWORD dwCookie
);
```

## Parameters

[in] dwCookie

Type: **DWORD**

Information about the "device removed" event, retrieved during a successful [RegisterDeviceRemovedEvent](#) call.

## Return value

None

## Remarks

See [RegisterDeviceRemovedEvent](#).

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d11_4.h
Library	D3d11.lib

## See also

[ID3D11Device4](#)

[RegisterDeviceRemovedEvent](#)

---

## Feedback

Was this page helpful?



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

# ID3D11Device5 interface (d3d11\_4.h)

Article07/27/2022

The device interface represents a virtual adapter; it is used to create resources. ID3D11Device5 adds new methods to those in [ID3D11Device4](#).

**Note** This interface, introduced in the Windows 10 Creators Update, is the latest version of the **ID3D11Device** interface. Applications targeting Windows 10 Creators Update should use this interface instead of earlier versions.

## Inheritance

The **ID3D11Device5** interface inherits from [ID3D11Device4](#). **ID3D11Device5** also has these types of members:

## Methods

The **ID3D11Device5** interface has these methods.

### [ID3D11Device5::CreateFence](#)

Creates a fence object. (`ID3D11Device5.CreateFence`)

### [ID3D11Device5::OpenSharedFence](#)

Opens a handle for a shared fence by using HANDLE and REFIID.

## Requirements

Target Platform

Windows

Header

d3d11\_4.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device5::CreateFence method (d3d11\_4.h)

Article 02/22/2024

Creates a fence object.

This member function is equivalent to the Direct3D 12 [ID3D12Device::CreateFence](#) member function, and applies between Direct3D 11 and Direct3D 12 in interop scenarios.

## Syntax

C++

```
HRESULT CreateFence(
    UINT64           InitialValue,
    D3D11_FENCE_FLAG Flags,
    REFIID           ReturnedInterface,
    [out] void       **ppFence
);
```

## Parameters

**InitialValue**

Type: [UINT64](#)

The initial value for the fence.

**Flags**

Type: [D3D11\\_FENCE\\_FLAG](#)

A combination of [D3D11\\_FENCE\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for the fence.

**ReturnedInterface**

Type: [REFIID](#)

The globally unique identifier ([GUID](#)) for the fence interface ([ID3D11Fence](#)). The [REFIID](#), or [GUID](#), of the interface to the fence can be obtained by using the `_uuidof()` macro.

For example, `_uuidof(ID3D11Fence)` will get the **GUID** of the interface to a fence.

[out] ppFence

Type: **void\*\***

A pointer to a memory block that receives a pointer to the [ID3D11Fence](#) interface that is used to access the fence.

## Return value

Type: [HRESULT](#)

Returns [S\\_OK](#) if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_4.h
Library	D3d11.lib

## See also

[ID3D11Device5](#)

[UnregisterDeviceRemoved](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Device5::OpenSharedFence method (d3d11\_4.h)

Article 02/22/2024

Opens a handle for a shared fence by using HANDLE and REFIID.

This member function is a limited version of the Direct3D 12 [ID3D12Device::OpenSharedHandle](#) member function, and applies between Direct3D 11 and Direct3D 12 in interop scenarios. Unlike [ID3D12Device::OpenSharedHandle](#) which operates on resources, heaps, and fences, the [ID3D11Device5::OpenSharedFence](#) function only operates on fences; in Direct3D 11, shared resources are opened with the [ID3D11Device::OpenSharedResource1](#) member function.

## Syntax

C++

```
HRESULT OpenSharedFence(
    [in]          HANDLE hFence,
    REFIID ReturnedInterface,
    [out, optional] void    **ppFence
);
```

## Parameters

[in] hFence

Type: **HANDLE**

The handle that was returned by a call to [ID3D11Fence::CreateSharedHandle](#) or [ID3D12Device::CreateSharedHandle](#).

ReturnedInterface

Type: **REFIID**

The globally unique identifier (**GUID**) for the [ID3D11Fence](#) interface. The **REFIID**, or **GUID**, of the interface can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D11Fence)` will get the **GUID** of the interface to the fence.

[out, optional] ppFence

Type: **void\*\***

A pointer to a memory block that receives a pointer to the [ID3D11Fence](#) interface.

## Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Requirements

  [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11_4.h
Library	D3d11.lib

## See also

[ID3D11Device5](#), [Multi-adapter systems](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceChild interface (d3d11.h)

Article07/27/2022

A device-child interface accesses data used by a device.

## Inheritance

The **ID3D11DeviceChild** interface inherits from the [IUnknown](#) interface.

**ID3D11DeviceChild** also has these types of members:

## Methods

The **ID3D11DeviceChild** interface has these methods.

<a href="#">ID3D11DeviceChild::GetDevice</a>
Get a pointer to the device that created this interface. ( <code>ID3D11DeviceChild.GetDevice</code> )
<a href="#">ID3D11DeviceChild::GetPrivateData</a>
Get application-defined data from a device child. ( <code>ID3D11DeviceChild.GetPrivateData</code> )
<a href="#">ID3D11DeviceChild::SetPrivateData</a>
Set application-defined data to a device child and associate that data with an application-defined guid. ( <code>ID3D11DeviceChild.SetPrivateData</code> )
<a href="#">ID3D11DeviceChild::SetPrivateDataInterface</a>
Associate an <code>IUnknown</code> -derived interface with this device child and associate that interface with an application-defined guid. ( <code>ID3D11DeviceChild.SetPrivateDataInterface</code> )

## Remarks

There are several types of device child interfaces, all of which inherit this interface. They include shaders, state objects, and input layouts.

**Windows Phone 8:** This API is supported.

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[IUnknown](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceChild::GetDevice method (d3d11.h)

Article 02/22/2024

Get a pointer to the device that created this interface.

## Syntax

C++

```
void GetDevice(  
    [out] ID3D11Device **ppDevice  
);
```

## Parameters

[out] ppDevice

Type: [ID3D11Device\\*\\*](#)

Address of a pointer to a device (see [ID3D11Device](#)).

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one, so be sure to call `::release()` on the returned pointer(s) before they are freed or else you will have a memory leak.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceChild::GetPrivateData method (d3d11.h)

Article 02/22/2024

Get application-defined data from a device child.

## Syntax

C++

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

## Parameters

[in] guid

Type: [REFGUID](#)

Guid associated with the data.

[in, out] pDataSize

Type: [UINT\\*](#)

A pointer to a variable that on input contains the size, in bytes, of the buffer that *pData* points to, and on output contains the size, in bytes, of the amount of data that *GetPrivateData* retrieved.

[out, optional] pData

Type: [void\\*](#)

A pointer to a buffer that *GetPrivateData* fills with data from the device child 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 11 Return Codes](#).

## Remarks

The data stored in the device child is set by calling [ID3D11DeviceChild::SetPrivateData](#).

If the data returned is a pointer to an [IUnknown](#), or one of its derivative classes, which was previously set by SetPrivateDataInterface, that interface will have its reference count incremented before the private data is returned.

**Windows Phone 8:** This API is supported.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceChild::SetPrivateData method (d3d11.h)

Article 07/27/2022

Set application-defined data to a device child and associate that data with an application-defined guid.

## Syntax

C++

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

## Parameters

[in] guid

Type: [REFGUID](#)

Guid associated with the data.

[in] DataSize

Type: [UINT](#)

Size of the data.

[in, optional] pData

Type: [const void\\*](#)

Pointer to the data to be stored with this device child. If pData is **NULL**, DataSize must also be 0, and any data previously associated with the specified guid will be destroyed.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

The data stored in the device child with this method can be retrieved with [ID3D11DeviceChild::GetPrivateData](#).

The [debug layer](#) reports memory leaks by outputting a list of object interface pointers along with their friendly names. The default friendly name is "<unnamed>". You can set the friendly name so that you can determine if the corresponding object interface pointer caused the leak. To set the friendly name, use the **SetPrivateData** method and the **WKP DID\_D3D Debug Object Name** GUID that is in D3Dcommon.h. For example, to give pContext a friendly name of *My name*, use the following code:

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

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceChild](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceChild::SetPrivateDataInterface method (d3d11.h)

Article 02/22/2024

Associate an IUnknown-derived interface with this device child and associate that interface with an application-defined guid.

## Syntax

C++

```
HRESULT SetPrivateDataInterface(
    [in]           REFGUID      guid,
    [in, optional] const IUnknown *pData
);
```

## Parameters

[in] guid

Type: [REFGUID](#)

Guid associated with the interface.

[in, optional] pData

Type: [const IUnknown\\*](#)

Pointer to an IUnknown-derived interface to be associated with the device child.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

When this method is called ::addref() will be called on the IUnknown-derived interface, and when the device child is destroyed ::release() will be called on the IUnknown-

derived interface.

# Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext interface (d3d11.h)

Article07/27/2022

The **ID3D11DeviceContext** interface represents a device context which generates rendering commands.

**Note** The latest version of this interface is **ID3D11DeviceContext4** introduced in the Windows 10 Creators Update. Applications targetting Windows 10 Creators Update should use the **ID3D11DeviceContext4** interface instead of **ID3D11DeviceContext**.

## Inheritance

The **ID3D11DeviceContext** interface inherits from [ID3D11DeviceChild](#).

**ID3D11DeviceContext** also has these types of members:

## Methods

The **ID3D11DeviceContext** interface has these methods.

### [ID3D11DeviceContext::Begin](#)

Mark the beginning of a series of commands.

### [ID3D11DeviceContext::ClearDepthStencilView](#)

Clears the depth-stencil resource. ([ID3D11DeviceContext.ClearDepthStencilView](#))

### [ID3D11DeviceContext::ClearRenderTargetView](#)

Set all the elements in a render target to one value.  
([ID3D11DeviceContext.ClearRenderTargetView](#))

### [ID3D11DeviceContext::ClearState](#)

Restore all default settings.

<a href="#">ID3D11DeviceContext::ClearUnorderedAccessViewFloat</a>	Clears an unordered access resource with a float value.
<a href="#">ID3D11DeviceContext::ClearUnorderedAccessViewUint</a>	Clears an unordered access resource with bit-precise values.
<a href="#">ID3D11DeviceContext::CopyResource</a>	Copy the entire contents of the source resource to the destination resource using the GPU. (ID3D11DeviceContext.CopyResource)
<a href="#">ID3D11DeviceContext::CopyStructureCount</a>	Copies data from a buffer holding variable length data.
<a href="#">ID3D11DeviceContext::CopySubresourceRegion</a>	Copy a region from a source resource to a destination resource. (ID3D11DeviceContext.CopySubresourceRegion)
<a href="#">ID3D11DeviceContext::CSGetConstantBuffers</a>	Get the constant buffers used by the compute-shader stage.
<a href="#">ID3D11DeviceContext::CSGetSamplers</a>	Get an array of sampler state interfaces from the compute-shader stage.
<a href="#">ID3D11DeviceContext::CSGetShader</a>	Get the compute shader currently set on the device.
<a href="#">ID3D11DeviceContext::CSGetShaderResources</a>	Get the compute-shader resources.
<a href="#">ID3D11DeviceContext::CSGetUnorderedAccessViews</a>	Gets an array of views for an unordered resource.
<a href="#">ID3D11DeviceContext::CSSetConstantBuffers</a>	Sets the constant buffers used by the compute-shader stage.
<a href="#">ID3D11DeviceContext::CSSetSamplers</a>	Set an array of sampler states to the compute-shader stage.

<a href="#">ID3D11DeviceContext::CSSetShader</a>
Set a compute shader to the device.
<a href="#">ID3D11DeviceContext::CSSetShaderResources</a>
Bind an array of shader resources to the compute-shader stage.
<a href="#">ID3D11DeviceContext::CSSetUnorderedAccessViews</a>
Sets an array of views for an unordered resource.
<a href="#">ID3D11DeviceContext::Dispatch</a>
Execute a command list from a thread group.
<a href="#">ID3D11DeviceContext::DispatchIndirect</a>
Execute a command list over one or more thread groups.
<a href="#">ID3D11DeviceContext::Draw</a>
Draw non-indexed, non-instanced primitives. ( <code>ID3D11DeviceContext.Draw</code> )
<a href="#">ID3D11DeviceContext::DrawAuto</a>
Draw geometry of an unknown size.
<a href="#">ID3D11DeviceContext::DrawIndexed</a>
Draw indexed, non-instanced primitives. ( <code>ID3D11DeviceContext.DrawIndexed</code> )
<a href="#">ID3D11DeviceContext::DrawIndexedInstanced</a>
Draw indexed, instanced primitives. ( <code>ID3D11DeviceContext.DrawIndexedInstanced</code> )
<a href="#">ID3D11DeviceContext::DrawIndexedInstancedIndirect</a>
Draw indexed, instanced, GPU-generated primitives.
<a href="#">ID3D11DeviceContext::DrawInstanced</a>
Draw non-indexed, instanced primitives. ( <code>ID3D11DeviceContext.DrawInstanced</code> )
<a href="#">ID3D11DeviceContext::DrawInstancedIndirect</a>
Draw instanced, GPU-generated primitives.

<p><a href="#">ID3D11DeviceContext::DSGetConstantBuffers</a></p> <p>Get the constant buffers used by the domain-shader stage.</p>
<p><a href="#">ID3D11DeviceContext::DSGetSamplers</a></p> <p>Get an array of sampler state interfaces from the domain-shader stage.</p>
<p><a href="#">ID3D11DeviceContext::DSGetShader</a></p> <p>Get the domain shader currently set on the device.</p>
<p><a href="#">ID3D11DeviceContext::DSGetShaderResources</a></p> <p>Get the domain-shader resources.</p>
<p><a href="#">ID3D11DeviceContext::DSSetConstantBuffers</a></p> <p>Sets the constant buffers used by the domain-shader stage.</p>
<p><a href="#">ID3D11DeviceContext::DSSetSamplers</a></p> <p>Set an array of sampler states to the domain-shader stage.</p>
<p><a href="#">ID3D11DeviceContext::DSSetShader</a></p> <p>Set a domain shader to the device.</p>
<p><a href="#">ID3D11DeviceContext::DSSetShaderResources</a></p> <p>Bind an array of shader resources to the domain-shader stage.</p>
<p><a href="#">ID3D11DeviceContext::End</a></p> <p>Mark the end of a series of commands.</p>
<p><a href="#">ID3D11DeviceContext::ExecuteCommandList</a></p> <p>Queues commands from a command list onto a device.</p>
<p><a href="#">ID3D11DeviceContext::FinishCommandList</a></p> <p>Create a command list and record graphics commands into it.</p>
<p><a href="#">ID3D11DeviceContext::Flush</a></p> <p>Sends queued-up commands in the command buffer to the graphics processing unit (GPU).</p>

<a href="#">ID3D11DeviceContext::GenerateMips</a>	Generates mipmaps for the given shader resource. (ID3D11DeviceContext.GenerateMips)
<a href="#">ID3D11DeviceContext::GetContextFlags</a>	Gets the initialization flags associated with the current deferred context.
<a href="#">ID3D11DeviceContext::GetData</a>	Get data from the graphics processing unit (GPU) asynchronously.
<a href="#">ID3D11DeviceContext::GetPredication</a>	Get the rendering predicate state. (ID3D11DeviceContext.GetPredication)
<a href="#">ID3D11DeviceContext::GetResourceMinLOD</a>	Gets the minimum level-of-detail (LOD).
<a href="#">ID3D11DeviceContext::GetType</a>	Gets the type of device context.
<a href="#">ID3D11DeviceContext::GSGetConstantBuffers</a>	Get the constant buffers used by the geometry shader pipeline stage. (ID3D11DeviceContext.GSGetConstantBuffers)
<a href="#">ID3D11DeviceContext::GSGetSamplers</a>	Get an array of sampler state interfaces from the geometry shader pipeline stage.
<a href="#">ID3D11DeviceContext::GSGetShader</a>	Get the geometry shader currently set on the device. (ID3D11DeviceContext.GSGetShader)
<a href="#">ID3D11DeviceContext::GSGetShaderResources</a>	Get the geometry shader resources. (ID3D11DeviceContext.GSGetShaderResources)
<a href="#">ID3D11DeviceContext::GSSetConstantBuffers</a>	Sets the constant buffers used by the geometry shader pipeline stage.
<a href="#">ID3D11DeviceContext::GSSetSamplers</a>	Set an array of sampler states to the geometry shader pipeline stage. (ID3D11DeviceContext.GSSetSamplers)

## [ID3D11DeviceContext::GSSetShader](#)

Set a geometry shader to the device. (`ID3D11DeviceContext.GSSetShader`)

## [ID3D11DeviceContext::GSSetShaderResources](#)

Bind an array of shader resources to the geometry shader stage.  
(`ID3D11DeviceContext.GSSetShaderResources`)

## [ID3D11DeviceContext::HSGetConstantBuffers](#)

Get the constant buffers used by the hull-shader stage.

## [ID3D11DeviceContext::HSGetSamplers](#)

Get an array of sampler state interfaces from the hull-shader stage.

## [ID3D11DeviceContext::HSGetShader](#)

Get the hull shader currently set on the device.

## [ID3D11DeviceContext::HSGetShaderResources](#)

Get the hull-shader resources.

## [ID3D11DeviceContext::HSSetConstantBuffers](#)

Set the constant buffers used by the hull-shader stage.

## [ID3D11DeviceContext::HSSetSamplers](#)

Set an array of sampler states to the hull-shader stage.

## [ID3D11DeviceContext::HSSetShader](#)

Set a hull shader to the device.

## [ID3D11DeviceContext::HSSetShaderResources](#)

Bind an array of shader resources to the hull-shader stage.

## [ID3D11DeviceContext::IAGetIndexBuffer](#)

Get a pointer to the index buffer that is bound to the input-assembler stage.  
(`ID3D11DeviceContext.IAGetIndexBuffer`)

## [ID3D11DeviceContext::IAGetInputLayout](#)

Get a pointer to the input-layout object that is bound to the input-assembler stage.  
([ID3D11DeviceContext.IAGetInputLayout](#))

## [ID3D11DeviceContext::IAGetPrimitiveTopology](#)

Get information about the primitive type, and data order that describes input data for the input assembler stage. ([ID3D11DeviceContext.IAGetPrimitiveTopology](#))

## [ID3D11DeviceContext::IAGetVertexBuffers](#)

Get the vertex buffers bound to the input-assembler stage.  
([ID3D11DeviceContext.IAGetVertexBuffers](#))

## [ID3D11DeviceContext::IASetIndexBuffer](#)

Bind an index buffer to the input-assembler stage. ([ID3D11DeviceContext.IASetIndexBuffer](#))

## [ID3D11DeviceContext::IASetInputLayout](#)

Bind an input-layout object to the input-assembler stage.  
([ID3D11DeviceContext.IASetInputLayout](#))

## [ID3D11DeviceContext::IASetPrimitiveTopology](#)

Bind information about the primitive type, and data order that describes input data for the input assembler stage. ([ID3D11DeviceContext.IASetPrimitiveTopology](#))

## [ID3D11DeviceContext::IASetVertexBuffers](#)

Bind an array of vertex buffers to the input-assembler stage.  
([ID3D11DeviceContext.IASetVertexBuffers](#))

## [ID3D11DeviceContext::Map](#)

Gets a pointer to the data contained in a subresource, and denies the GPU access to that subresource.

## [ID3D11DeviceContext::OMGetBlendState](#)

Get the blend state of the output-merger stage. ([ID3D11DeviceContext.OMGetBlendState](#))

## [ID3D11DeviceContext::OMGetDepthStencilState](#)

Gets the depth-stencil state of the output-merger stage.  
([ID3D11DeviceContext.OMGetDepthStencilState](#))

## [ID3D11DeviceContext::OMGetRenderTarget](#)

Get pointers to the resources bound to the output-merger stage.  
(ID3D11DeviceContext.OMGetRenderTarget)

## [ID3D11DeviceContext::OMGetRenderTargetAndUnorderedAccessViews](#)

Get pointers to the resources bound to the output-merger stage.  
(ID3D11DeviceContext.OMGetRenderTargetAndUnorderedAccessViews)

## [ID3D11DeviceContext::OMSetBlendState](#)

Set the blend state of the output-merger stage. (ID3D11DeviceContext.OMSetBlendState)

## [ID3D11DeviceContext::OMSetDepthStencilState](#)

Sets the depth-stencil state of the output-merger stage.  
(ID3D11DeviceContext.OMSetDepthStencilState)

## [ID3D11DeviceContext::OMSetRenderTarget](#)

Bind one or more render targets atomically and the depth-stencil buffer to the output-merger stage.

## [ID3D11DeviceContext::OMSetRenderTargetAndUnorderedAccessViews](#)

Binds resources to the output-merger stage.

## [ID3D11DeviceContext::PSGetConstantBuffers](#)

Get the constant buffers used by the pixel shader pipeline stage.  
(ID3D11DeviceContext.PSGetConstantBuffers)

## [ID3D11DeviceContext::PSGetSamplers](#)

Get an array of sampler states from the pixel shader pipeline stage.  
(ID3D11DeviceContext.PSGetSamplers)

## [ID3D11DeviceContext::PSGetShader](#)

Get the pixel shader currently set on the device. (ID3D11DeviceContext.PSGetShader)

## [ID3D11DeviceContext::PSGetShaderResources](#)

Get the pixel shader resources. (ID3D11DeviceContext.PSGetShaderResources)

## [ID3D11DeviceContext::PSSetConstantBuffers](#)

Sets the constant buffers used by the pixel shader pipeline stage.

<a href="#">ID3D11DeviceContext::PSSetSamplers</a>
Set an array of sampler states to the pixel shader pipeline stage. (ID3D11DeviceContext.PSSetSamplers)
<a href="#">ID3D11DeviceContext::PSSetShader</a>
Sets a pixel shader to the device. (ID3D11DeviceContext.PSSetShader)
<a href="#">ID3D11DeviceContext::PSSetShaderResources</a>
Bind an array of shader resources to the pixel shader stage. (ID3D11DeviceContext.PSSetShaderResources)
<a href="#">ID3D11DeviceContext::ResolveSubresource</a>
Copy a multisampled resource into a non-multisampled resource.
<a href="#">ID3D11DeviceContext::RSGetScissorRects</a>
Get the array of scissor rectangles bound to the rasterizer stage. (ID3D11DeviceContext.RSGetScissorRects)
<a href="#">ID3D11DeviceContext::RSGetState</a>
Get the rasterizer state from the rasterizer stage of the pipeline. (ID3D11DeviceContext.RSGetState)
<a href="#">ID3D11DeviceContext::RSGetViewports</a>
Gets the array of viewports bound to the rasterizer stage.
<a href="#">ID3D11DeviceContext::RSSetScissorRects</a>
Bind an array of scissor rectangles to the rasterizer stage. (ID3D11DeviceContext.RSSetScissorRects)
<a href="#">ID3D11DeviceContext::RSSetState</a>
Set the rasterizer state for the rasterizer stage of the pipeline. (ID3D11DeviceContext.RSSetState)
<a href="#">ID3D11DeviceContext::RSSetViewports</a>
Bind an array of viewports to the rasterizer stage of the pipeline. (ID3D11DeviceContext.RSSetViewports)
<a href="#">ID3D11DeviceContext::SetPredication</a>
Set a rendering predicate. (ID3D11DeviceContext.SetPredication)

## [ID3D11DeviceContext::SetResourceMinLOD](#)

Sets the minimum level-of-detail (LOD) for a resource.

## [ID3D11DeviceContext::SOGetTargets](#)

Get the target output buffers for the stream-output stage of the pipeline.

## [ID3D11DeviceContext::SOSetTargets](#)

Set the target output buffers for the stream-output stage of the pipeline.

## [ID3D11DeviceContext::Unmap](#)

Invalidate the pointer to a resource and reenable the GPU's access to that resource.

## [ID3D11DeviceContext::UpdateSubresource](#)

The CPU copies data from memory to a subresource created in non-mappable memory.  
([ID3D11DeviceContext.UpdateSubresource](#))

## [ID3D11DeviceContext::VSGetConstantBuffers](#)

Get the constant buffers used by the vertex shader pipeline stage.  
([ID3D11DeviceContext.VSGetConstantBuffers](#))

## [ID3D11DeviceContext::VSGetSamplers](#)

Get an array of sampler states from the vertex shader pipeline stage.  
([ID3D11DeviceContext.VSGetSamplers](#))

## [ID3D11DeviceContext::VSGetShader](#)

Get the vertex shader currently set on the device. ([ID3D11DeviceContext.VSGetShader](#))

## [ID3D11DeviceContext::VSGetShaderResources](#)

Get the vertex shader resources. ([ID3D11DeviceContext.VSGetShaderResources](#))

## [ID3D11DeviceContext::VSSetConstantBuffers](#)

Sets the constant buffers used by the vertex shader pipeline stage.

## [ID3D11DeviceContext::VSSetSamplers](#)

Set an array of sampler states to the vertex shader pipeline stage.  
([ID3D11DeviceContext.VSSetSamplers](#))

## [ID3D11DeviceContext::VSSetShader](#)

Set a vertex shader to the device. ([ID3D11DeviceContext.VSSetShader](#))

## [ID3D11DeviceContext::VSSetShaderResources](#)

Bind an array of shader resources to the vertex-shader stage.

# Requirements

<b>Minimum supported client</b>	Windows 7 [desktop apps   UWP apps]
<b>Minimum supported server</b>	Windows Server 2008 R2 [desktop apps   UWP apps]
<b>Target Platform</b>	Windows
<b>Header</b>	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::Begin method (d3d11.h)

Article 02/22/2024

Mark the beginning of a series of commands.

## Syntax

C++

```
void Begin(  
    [in] ID3D11Asynchronous *pAsync  
);
```

## Parameters

[in] pAsync

Type: [ID3D11Asynchronous\\*](#)

A pointer to an [ID3D11Asynchronous](#) interface.

## Return value

None

## Remarks

Use [ID3D11DeviceContext::End](#) to mark the ending of the series of commands.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::ClearDepthStencilView method (d3d11.h)

Article 07/27/2022

Clears the depth-stencil resource.

## Syntax

C++

```
void ClearDepthStencilView(
    [in] ID3D11DepthStencilView *pDepthStencilView,
    [in] UINT                 ClearFlags,
    [in] FLOAT                Depth,
    [in] UINT8               Stencil
);
```

## Parameters

[in] pDepthStencilView

Type: [ID3D11DepthStencilView\\*](#)

Pointer to the depth stencil to be cleared.

[in] ClearFlags

Type: [UINT](#)

Identify the type of data to clear (see [D3D11\\_CLEAR\\_FLAG](#)).

[in] Depth

Type: [FLOAT](#)

Clear the depth buffer with this value. This value will be clamped between 0 and 1.

[in] Stencil

Type: [UINT8](#)

Clear the stencil buffer with this value.

# Return value

None

## Remarks

Differences between Direct3D 9 and Direct3D 11/10:

Unlike Direct3D 9, the full extent of the resource view is always cleared. Viewport and scissor settings are not applied.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::ClearRenderTargetView method (d3d11.h)

Article 02/22/2024

Set all the elements in a render target to one value.

## Syntax

C++

```
void ClearRenderTargetView(  
    [in] ID3D11RenderTargetView *pRenderTargetView,  
    [in] const FLOAT [4]           ColorRGBA  
) ;
```

## Parameters

[in] pRenderTargetView

Type: [ID3D11RenderTargetView\\*](#)

Pointer to the render target.

[in] ColorRGBA

Type: [const FLOAT\[4\]](#)

A 4-component array that represents the color to fill the render target with.

## Return value

None

## Remarks

Applications that wish to clear a render target to a specific integer value bit pattern should render a screen-aligned quad instead of using this method. The reason for this is because this method accepts as input a floating point value, which may not have the same bit pattern as the original integer.

[ ] Expand table

Differences between Direct3D 9 and Direct3D 11/10:

Unlike Direct3D 9, the full extent of the resource view is always cleared. Viewport and scissor settings are not applied.

When using `D3D_FEATURE_LEVEL_9_x`, `ClearRenderTargetView` only clears the first array slice in the render target view. This can impact (for example) cube map rendering scenarios. Applications should create a render target view for each face or array slice, then clear each view individually.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	<code>d3d11.h</code>
Library	<code>D3D11.lib</code>

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::ClearState method (d3d11.h)

Article 02/22/2024

Restore all default settings.

## Syntax

C++

```
void ClearState();
```

## Return value

None

## Remarks

This method resets any device context to the default settings. This sets all input/output resource slots, shaders, input layouts, predication, scissor rectangles, depth-stencil state, rasterizer state, blend state, sampler state, and viewports to **NULL**. The primitive topology is set to **UNDEFINED**.

For a scenario where you would like to clear a list of commands recorded so far, call [ID3D11DeviceContext::FinishCommandList](#) and throw away the resulting [ID3D11CommandList](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::ClearUnorderedAccessViewUint method (d3d11.h)

Article 02/22/2024

Clears an [unordered access](#) resource with bit-precise values.

## Syntax

C++

```
void ClearUnorderedAccessViewUint(
    [in] ID3D11UnorderedAccessView *pUnorderedAccessView,
    [in] const UINT [4]           Values
);
```

## Parameters

[in] pUnorderedAccessView

Type: [ID3D11UnorderedAccessView\\*](#)

The [ID3D11UnorderedAccessView](#) to clear.

[in] Values

Type: [const UINT\[4\]](#)

Values to copy to corresponding channels, see remarks.

## Return value

None

## Remarks

This API copies the lower  $n_i$  bits from each array element  $i$  to the corresponding channel, where  $n_i$  is the number of bits in the  $i$ th channel of the resource format (for example, R8G8B8\_FLOAT has 8 bits for the first 3 channels). This works on any UAV with no format conversion. For a raw or structured buffer view, only the first array element value is used.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::ClearUnorderedAccessViewFloat method (d3d11.h)

Article 02/22/2024

Clears an [unordered access](#) resource with a float value.

## Syntax

C++

```
void ClearUnorderedAccessViewFloat(
    [in] ID3D11UnorderedAccessView *pUnorderedAccessView,
    [in] const FLOAT [4]           Values
);
```

## Parameters

[in] pUnorderedAccessView

Type: [ID3D11UnorderedAccessView\\*](#)

The [ID3D11UnorderedAccessView](#) to clear.

[in] Values

Type: [const FLOAT\[4\]](#)

Values to copy to corresponding channels, see remarks.

## Return value

None

## Remarks

This API works on FLOAT, UNORM, and SNORM unordered access views (UAVs), with format conversion from FLOAT to \*NORM where appropriate. On other UAVs, the operation is invalid and the call will not reach the driver.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CopyResource method (d3d11.h)

Article 07/27/2022

Copy the entire contents of the source resource to the destination resource using the GPU.

## Syntax

C++

```
void CopyResource(  
    [in] ID3D11Resource *pDstResource,  
    [in] ID3D11Resource *pSrcResource  
);
```

## Parameters

[in] pDstResource

Type: [ID3D11Resource\\*](#)

A pointer to the [ID3D11Resource](#) interface that represents the destination resource.

[in] pSrcResource

Type: [ID3D11Resource\\*](#)

A pointer to the [ID3D11Resource](#) interface that represents the source resource.

## Return value

None

## Remarks

This method is unusual in that it causes the GPU to perform the copy operation (similar to a `memcpy` by the CPU). As a result, it 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 a `DXGI_FORMAT_R32G32B32_UINT` texture since both of these formats are in the `DXGI_FORMAT_R32G32B32_TYPELESS` group. [CopyResource](#) can copy between a few format types. 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 `D3D11_BIND_DEPTH_STENCIL` flag can only be used as a source for [CopyResource](#). Resources created with multisampling capability (see [DXGI\\_SAMPLE\\_DESC](#)) can be used as source and destination only if both source and destination have identical multisampled count and quality. If source and destination differ in multisampled count and quality or if one is multisampled and the other is not multisampled, the call to [ID3D11DeviceContext::CopyResource](#) fails. Use [ID3D11DeviceContext::ResolveSubresource](#) to resolve a multisampled resource to a resource that is not multisampled.

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](#).

We recommend to use [ID3D11DeviceContext::CopySubresourceRegion](#) instead if you only need to copy a portion of the data in a resource.

## Requirements

Target Platform	Windows
Header	d3d11.h

Library

D3D11.lib

## See also

[ID3D11DeviceContext](#)

[ID3D11Resource](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CopyStructureCount method (d3d11.h)

Article 02/22/2024

Copies data from a buffer holding variable length data.

## Syntax

C++

```
void CopyStructureCount(
    [in] ID3D11Buffer          *pDstBuffer,
    [in] UINT                  DstAlignedByteOffset,
    [in] ID3D11UnorderedAccessView *pSrcView
);
```

## Parameters

[in] pDstBuffer

Type: [ID3D11Buffer\\*](#)

Pointer to [ID3D11Buffer](#). This can be any buffer resource that other copy commands, such as [ID3D11DeviceContext::CopyResource](#) or [ID3D11DeviceContext::CopySubresourceRegion](#), are able to write to.

[in] DstAlignedByteOffset

Type: [UINT](#)

Offset from the start of *pDstBuffer* to write 32-bit [UINT](#) structure (vertex) count from *pSrcView*.

[in] pSrcView

Type: [ID3D11UnorderedAccessView\\*](#)

Pointer to an [ID3D11UnorderedAccessView](#) of a Structured Buffer resource created with either [D3D11\\_BUFFER\\_UAV\\_FLAG\\_APPEND](#) or [D3D11\\_BUFFER\\_UAV\\_FLAG\\_COUNTER](#) specified when the UAV was created. These types of resources have hidden counters tracking "how many" records have been written.

# Return value

None

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CopySubresourceRegion method (d3d11.h)

Article 11/18/2024

Copy a region from a source resource to a destination resource.

## Syntax

C++

```
void CopySubresourceRegion(
    [in]          ID3D11Resource  *pDstResource,
    [in]          UINT           DstSubresource,
    [in]          UINT           DstX,
    [in]          UINT           DstY,
    [in]          UINT           DstZ,
    [in]          ID3D11Resource  *pSrcResource,
    [in]          UINT           SrcSubresource,
    [in, optional] const D3D11_BOX *pSrcBox
);
```

## Parameters

[in] pDstResource

Type: [ID3D11Resource\\*](#)

A pointer to the destination resource (see [ID3D11Resource](#)).

[in] DstSubresource

Type: [UINT](#)

Destination subresource index.

[in] DstX

Type: [UINT](#)

The x-coordinate of the upper left corner of the destination region.

[in] DstY

Type: [UINT](#)

The y-coordinate of the upper left corner of the destination region. For a 1D subresource, this must be zero.

[in] DstZ

Type: **UINT**

The z-coordinate of the upper left corner of the destination region. For a 1D or 2D subresource, this must be zero.

[in] pSrcResource

Type: **ID3D11Resource\***

A pointer to the source resource (see [ID3D11Resource](#)).

[in] SrcSubresource

Type: **UINT**

Source subresource index.

[in, optional] pSrcBox

Type: **const D3D11\_BOX\***

A pointer to a 3D box (see [D3D11\\_BOX](#)) that defines the source subresource that can be copied. If **NULL**, the entire source subresource is copied. The box must fit within the source resource.

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, [CopySubresourceRegion](#) doesn't perform a copy operation.

## 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 **CopySubresourceRegion** is undefined. If you created a device that supports the [debug layer](#), the debug output reports an error on this invalid **CopySubresourceRegion** call. Invalid parameters to **CopySubresourceRegion** 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. [D3D11CalcSubresource](#) is a helper function for calculating subresource indexes.

**CopySubresourceRegion** 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 be the same type.
- Must have compatible DXGI formats (identical or from the same type group). For example, a `DXGI_FORMAT_R32G32B32_FLOAT` texture can be copied to a `DXGI_FORMAT_R32G32B32_UINT` texture since both of these formats are in the `DXGI_FORMAT_R32G32B32_TYPELESS` group. **CopySubresourceRegion** can copy between a few format types. For more info, see [Format Conversion using Direct3D 10.1](#).
- May not be currently mapped.

**CopySubresourceRegion** only supports copy; it doesn't support any stretch, color key, or blend. **CopySubresourceRegion** can reinterpret the resource data between a few format types. For more info, see [Format conversion using Direct3D 10.1](#).

If your app needs to copy an entire resource, we recommend to use [ID3D11DeviceContext::CopyResource](#) instead.

**CopySubresourceRegion** 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 information about pipeline stalls, see [performance considerations](#).

**Note** Applies only to feature level 9\_x hardware If you use [ID3D11DeviceContext::UpdateSubresource](#) or **CopySubresourceRegion** to copy from a staging resource to a default resource, you can corrupt the destination contents. This occurs if you pass a **NULL** source box and if the source resource has different dimensions from those of the destination resource or if you use

destination offsets, (x, y, and z). In this situation, always pass a source box that is the full size of the source resource.

**Note** Applies only to feature level 9\_x hardware You can't use **CopySubresourceRegion** to copy mipmapped volume textures.

**Note** Applies only to feature levels 9\_x Subresources created with the D3D11\_BIND\_DEPTH\_STENCIL flag can only be used as a source for **CopySubresourceRegion**.

**Note** If you use **CopySubresourceRegion** with a depth-stencil buffer or a multisampled resource, you must copy the whole subresource. 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.

## Example

The following code snippet copies a box (located at (120,100),(200,220)) from a source texture into a region (10,20),(90,140) in a destination texture.

```
D3D11_BOX sourceRegion;
sourceRegion.left = 120;
sourceRegion.right = 200;
sourceRegion.top = 100;
sourceRegion.bottom = 220;
sourceRegion.front = 0;
sourceRegion.back = 1;

pd3dDeviceContext->CopySubresourceRegion( pDestTexture, 0, 10, 20, 0,
pSourceTexture, 0, &sourceRegion );
```

Notice, that for a 2D texture, front and back are set to 0 and 1 respectively.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

[ID3D11Resource](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSGetConstantBuffers method (d3d11.h)

Article 02/22/2024

Get the constant buffers used by the compute-shader stage.

## Syntax

C++

```
void CSGetConstantBuffers(  
    [in]          UINT      StartSlot,  
    [in]          UINT      NumBuffers,  
    [out, optional] ID3D11Buffer **ppConstantBuffers  
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - StartSlot).

[out, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*\\*](#)

Array of constant buffer interface pointers (see [ID3D11Buffer](#)) to be returned by the method.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSGetSamplers method (d3d11.h)

Article 02/22/2024

Get an array of sampler state interfaces from the compute-shader stage.

## Syntax

C++

```
void CSGetSamplers(
    [in]           UINT          StartSlot,
    [in]           UINT          NumSamplers,
    [out, optional] ID3D11SamplerState **ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into a zero-based array to begin getting samplers from (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers to get from a device context. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[out, optional] ppSamplers

Type: [ID3D11SamplerState\\*\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)).

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSGetShader method (d3d11.h)

Article 02/22/2024

Get the compute shader currently set on the device.

## Syntax

C++

```
void CSGetShader(
    [out]           ID3D11ComputeShader **ppComputeShader,
    [out, optional] ID3D11ClassInstance **ppClassInstances,
    [in, out, optional] UINT             *pNumClassInstances
);
```

## Parameters

[out] ppComputeShader

Type: [ID3D11ComputeShader\\*\\*](#)

Address of a pointer to a Compute shader (see [ID3D11ComputeShader](#)) to be returned by the method.

[out, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*\\*](#)

Pointer to an array of class instance interfaces (see [ID3D11ClassInstance](#)).

[in, out, optional] pNumClassInstances

Type: [UINT\\*](#)

The number of class-instance elements in the array.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSGetShaderResources method (d3d11.h)

Article 02/22/2024

Get the compute-shader resources.

## Syntax

C++

```
void CSGetShaderResources(
    [in]          UINT           StartSlot,
    [in]          UINT           NumViews,
    [out, optional] ID3D11ShaderResourceView **ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin getting shader resources from (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

The number of resources to get from the device. Up to a maximum of 128 slots are available for shader resources (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[out, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*\\*](#)

Array of [shader resource view](#) interfaces to be returned by the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSGetUnorderedAccessViews method (d3d11.h)

Article 02/22/2024

Gets an array of views for an unordered resource.

## Syntax

C++

```
void CSGetUnorderedAccessViews(
    [in]          UINT             StartSlot,
    [in]          UINT             NumUAVs,
    [out, optional] ID3D11UnorderedAccessView **ppUnorderedAccessViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index of the first element in the zero-based array to return (ranges from 0 to D3D11\_1\_UAV\_SLOT\_COUNT - 1).

[in] NumUAVs

Type: [UINT](#)

Number of views to get (ranges from 0 to D3D11\_1\_UAV\_SLOT\_COUNT - StartSlot).

[out, optional] ppUnorderedAccessViews

Type: [ID3D11UnorderedAccessView\\*\\*](#)

A pointer to an array of interface pointers (see [ID3D11UnorderedAccessView](#)) to get.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSSetConstantBuffers method (d3d11.h)

Article 02/22/2024

Sets the constant buffers used by the compute-shader stage.

## Syntax

C++

```
void CSSetConstantBuffers(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the zero-based array to begin setting constant buffers to (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to set (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - *StartSlot*).

[in, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*](#)

Array of constant buffers (see [ID3D11Buffer](#)) being given to the device.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The Direct3D 11.1 runtime, which is available starting with Windows 8, can bind a larger number of [ID3D11Buffer](#) resources to the shader than the maximum constant buffer size that is supported by shaders (4096 constants – 4\*32-bit components each). When you bind such a large buffer, the shader can access only the first 4096 4\*32-bit component constants in the buffer, as if 4096 constants is the full size of the buffer.

If the application wants the shader to access other parts of the buffer, it must call the [CSSetConstantBuffers1](#) method instead.

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSSetSamplers method (d3d11.h)

Article 02/22/2024

Set an array of sampler states to the compute-shader stage.

## Syntax

C++

```
void CSSetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [in, optional] ID3D11SamplerState * const *ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting samplers to (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers in the array. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[in, optional] ppSamplers

Type: [ID3D11SamplerState\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)). See Remarks.

## Return value

None

## Remarks

Any sampler may be set to **NULL**; this invokes the default state, which is defined to be the following.

```
//Default sampler state:  
D3D11_SAMPLER_DESC SamplerDesc;  
SamplerDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;  
SamplerDesc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.MipLODBias = 0;  
SamplerDesc.MaxAnisotropy = 1;  
SamplerDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;  
SamplerDesc.BorderColor[0] = 1.0f;  
SamplerDesc.BorderColor[1] = 1.0f;  
SamplerDesc.BorderColor[2] = 1.0f;  
SamplerDesc.BorderColor[3] = 1.0f;  
SamplerDesc.MinLOD = -FLT_MAX;  
SamplerDesc.MaxLOD = FLT_MAX;
```

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSSetShader method (d3d11.h)

Article 02/22/2024

Set a compute shader to the device.

## Syntax

C++

```
void CSSetShader(  
    [in, optional] ID3D11ComputeShader *pComputeShader,  
    [in, optional] ID3D11ClassInstance * const *ppClassInstances,  
    UINT NumClassInstances  
) ;
```

## Parameters

[in, optional] pComputeShader

Type: [ID3D11ComputeShader\\*](#)

Pointer to a compute shader (see [ID3D11ComputeShader](#)). Passing in **NULL** disables the shader for this pipeline stage.

[in, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*](#)

A pointer to an array of class-instance interfaces (see [ID3D11ClassInstance](#)). Each interface used by a shader must have a corresponding class instance or the shader will get disabled. Set ppClassInstances to **NULL** if the shader does not use any interfaces.

NumClassInstances

Type: [UINT](#)

The number of class-instance interfaces in the array.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The maximum number of instances a shader can have is 256.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSSetShaderResources method (d3d11.h)

Article 02/22/2024

Bind an array of shader resources to the compute-shader stage.

## Syntax

C++

```
void CSSetShaderResources(
    [in]          UINT             StartSlot,
    [in]          UINT             NumViews,
    [in, optional] ID3D11ShaderResourceView * const *ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting shader resources to (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

Number of shader resources to set. Up to a maximum of 128 slots are available for shader resources(ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[in, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*](#)

Array of [shader resource view](#) interfaces to set to the device.

## Return value

None

## Remarks

If an overlapping resource view is already bound to an output slot, such as a render target, then the method will fill the destination shader resource slot with **NULL**.

For information about creating shader-resource views, see [ID3D11Device::CreateShaderResourceView](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::CSSetUnorderedAccessViews method (d3d11.h)

Article 10/13/2021

Sets an array of views for an unordered resource.

## Syntax

C++

```
void CSSetUnorderedAccessViews(
    [in]          UINT             StartSlot,
    [in]          UINT             NumUAVs,
    [in, optional] ID3D11UnorderedAccessView * const *ppUnorderedAccessViews,
    [in, optional] const UINT        *pUAVInitialCounts
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index of the first element in the zero-based array to begin setting (ranges from 0 to D3D11\_1\_UAV\_SLOT\_COUNT - 1). D3D11\_1\_UAV\_SLOT\_COUNT is defined as 64.

[in] NumUAVs

Type: [UINT](#)

Number of views to set (ranges from 0 to D3D11\_1\_UAV\_SLOT\_COUNT - *StartSlot*).

[in, optional] ppUnorderedAccessViews

Type: [ID3D11UnorderedAccessView\\*](#)

A pointer to an array of [ID3D11UnorderedAccessView](#) pointers to be set by the method.

[in, optional] pUAVInitialCounts

Type: [const UINT\\*](#)

An array of append and consume buffer offsets. A value of -1 indicates to keep the current offset. Any other values set the hidden counter for that appendable and consumable UAV. *pUAVInitialCounts* is only relevant for UAVs that were created with either **D3D11\_BUFFER\_UAV\_FLAG\_APPEND** or **D3D11\_BUFFER\_UAV\_FLAG\_COUNTER** specified when the UAV was created; otherwise, the argument is ignored.

## Return value

None

## Remarks

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::Dispatch method (d3d11.h)

Article 02/22/2024

Execute a command list from a thread group.

## Syntax

C++

```
void Dispatch(
    [in] UINT ThreadGroupCountX,
    [in] UINT ThreadGroupCountY,
    [in] UINT ThreadGroupCountZ
);
```

## Parameters

[in] ThreadGroupCountX

Type: **UINT**

The number of groups dispatched in the x direction. *ThreadGroupCountX* must be less than or equal to D3D11\_CS\_DISPATCH\_MAX\_THREAD\_GROUPS\_PER\_DIMENSION (65535).

[in] ThreadGroupCountY

Type: **UINT**

The number of groups dispatched in the y direction. *ThreadGroupCountY* must be less than or equal to D3D11\_CS\_DISPATCH\_MAX\_THREAD\_GROUPS\_PER\_DIMENSION (65535).

[in] ThreadGroupCountZ

Type: **UINT**

The number of groups dispatched in the z direction. *ThreadGroupCountZ* must be less than or equal to D3D11\_CS\_DISPATCH\_MAX\_THREAD\_GROUPS\_PER\_DIMENSION (65535). In feature level 10 the value for *ThreadGroupCountZ* must be 1.

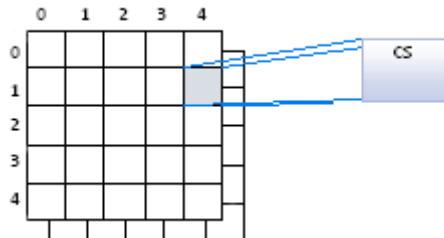
# Return value

None

## Remarks

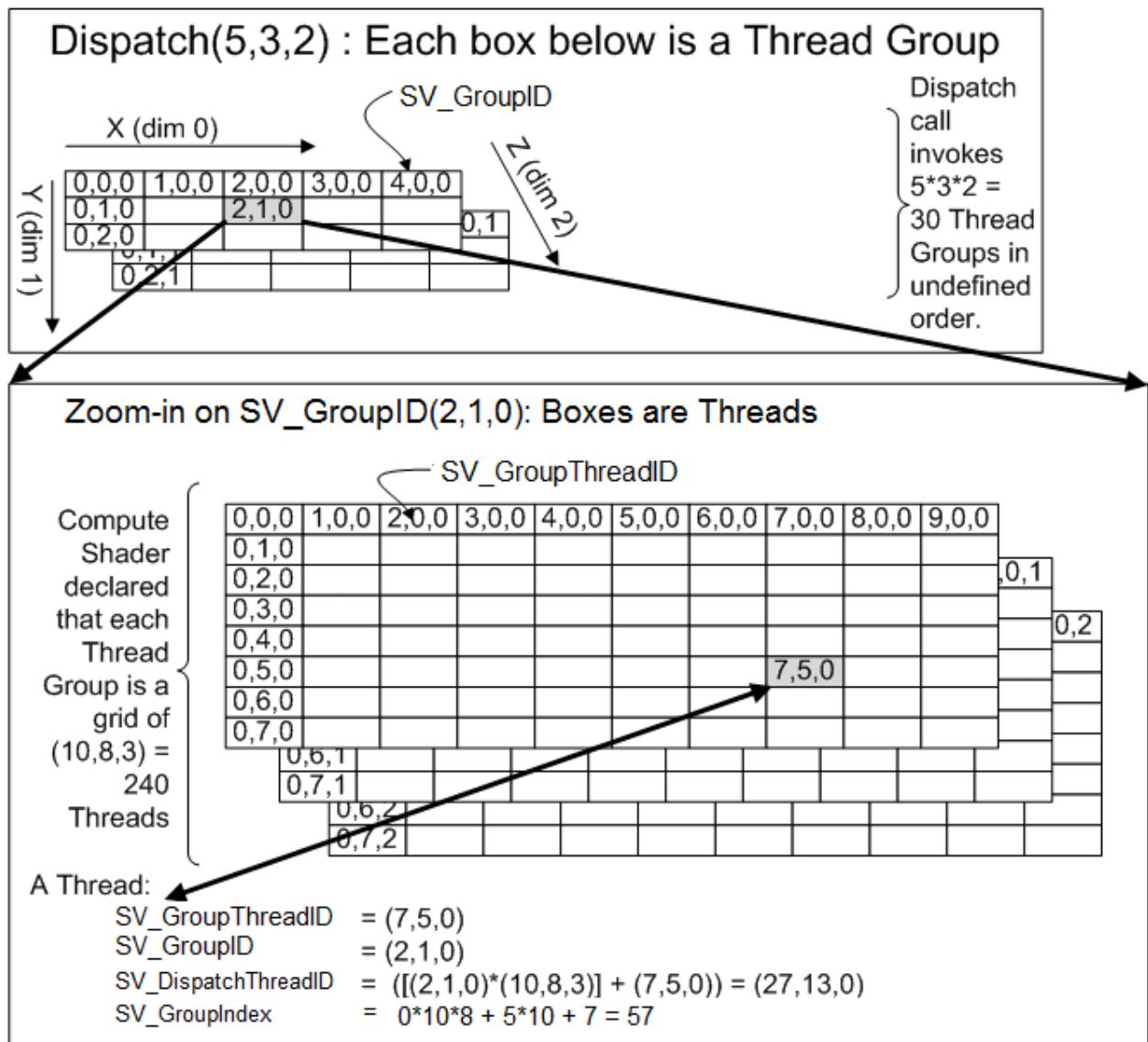
You call the **Dispatch** method to execute commands in a [compute shader](#). A compute shader can be run on many threads in parallel, within a thread group. Index a particular thread, within a thread group using a 3D vector given by (x,y,z).

In the following illustration, assume a thread group with 50 threads where the size of the group is given by (5,5,2). A single thread is identified from a thread group with 50 threads in it, using the vector (4,1,1).



The following illustration shows the relationship between the parameters passed to **ID3D11DeviceContext::Dispatch**, Dispatch(5,3,2), the values specified in the [numthreads](#) attribute, numthreads(10,8,3), and values that will be passed to the compute shader for the thread-related system values

([SV\\_GroupIndex](#),[SV\\_DispatchThreadID](#),[SV\\_GroupThreadID](#),[SV\\_GroupID](#)).



## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DispatchIndirect method (d3d11.h)

Article 02/22/2024

Execute a command list over one or more thread groups.

## Syntax

C++

```
void DispatchIndirect(
    [in] ID3D11Buffer *pBufferForArgs,
    [in] UINT          AlignedByteOffsetForArgs
);
```

## Parameters

[in] pBufferForArgs

Type: [ID3D11Buffer\\*](#)

A pointer to an [ID3D11Buffer](#), which must be loaded with data that matches the argument list for [ID3D11DeviceContext::Dispatch](#).

[in] AlignedByteOffsetForArgs

Type: [UINT](#)

A byte-aligned offset between the start of the buffer and the arguments.

## Return value

None

## Remarks

You call the [DispatchIndirect](#) method to execute commands in a [compute shader](#).

When an application creates a buffer that is associated with the [ID3D11Buffer](#) interface that *pBufferForArgs* points to, the application must set the

`D3D11_RESOURCE_MISC_DRAWINDIRECT_ARGS` flag in the `MiscFlags` member of the `D3D11_BUFFER_DESC` structure that describes the buffer. To create the buffer, the application calls the `ID3D11Device::CreateBuffer` method and in this call passes a pointer to `D3D11_BUFFER_DESC` in the *pDesc* parameter.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::Draw method (d3d11.h)

Article 02/22/2024

Draw non-indexed, non-instanced primitives.

## Syntax

C++

```
void Draw(
    [in] UINT VertexCount,
    [in] UINT StartVertexLocation
);
```

## Parameters

[in] VertexCount

Type: **UINT**

Number of vertices to draw.

[in] StartVertexLocation

Type: **UINT**

Index of the first vertex, which is usually an offset in a vertex buffer.

## Return value

None

## Remarks

Draw submits work to the rendering pipeline.

The vertex data for a draw call normally comes from a vertex buffer that is bound to the pipeline.

Even without any vertex buffer bound to the pipeline, you can generate your own vertex data in your vertex shader by using the [SV\\_VertexID](#) system-value semantic to determine the current vertex that the runtime is processing.

## Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DrawAuto method (d3d11.h)

Article 06/29/2021

Draw geometry of an unknown size.

## Syntax

C++

```
void DrawAuto();
```

## Return value

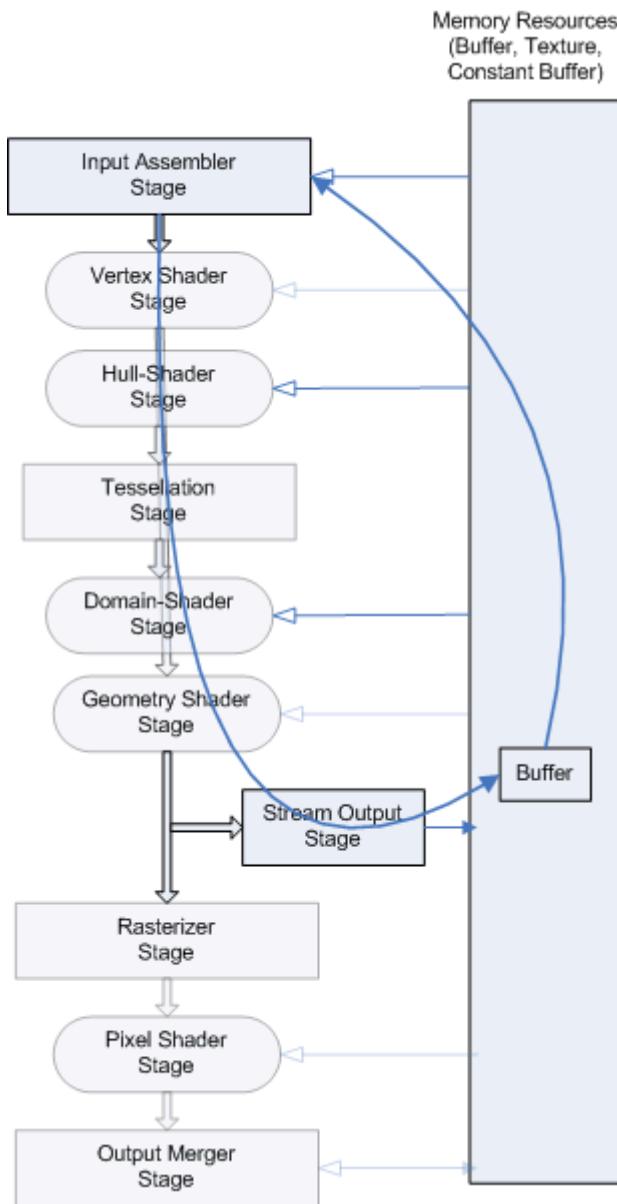
None

## Remarks

A draw API submits work to the rendering pipeline. This API submits work of an unknown size that was processed by the input assembler, vertex shader, and stream-output stages; the work may or may not have gone through the geometry-shader stage.

After data has been streamed out to stream-output stage buffers, those buffers can be again bound to the Input Assembler stage at input slot 0 and DrawAuto will draw them without the application needing to know the amount of data that was written to the buffers. A measurement of the amount of data written to the SO stage buffers is maintained internally when the data is streamed out. This means that the CPU does not need to fetch the measurement before re-binding the data that was streamed as input data. Although this amount is tracked internally, it is still the responsibility of applications to use input layouts to describe the format of the data in the SO stage buffers so that the layouts are available when the buffers are again bound to the input assembler.

The following diagram shows the DrawAuto process.



Calling `DrawAuto` does not change the state of the streaming-output buffers that were bound again as inputs.

`DrawAuto` only works when drawing with one input buffer bound as an input to the IA stage at slot 0. Applications must create the SO buffer resource with both binding flags, [D3D11\\_BIND\\_VERTEX\\_BUFFER](#) and [D3D11\\_BIND\\_STREAM\\_OUTPUT](#).

This API does not support indexing or instancing.

If an application needs to retrieve the size of the streaming-output buffer, it can query for statistics on streaming output by using [D3D11\\_QUERY\\_SO\\_STATISTICS](#).

## Requirements

Target Platform	Windows
-----------------	---------

Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DrawIndexed method (d3d11.h)

Article 02/22/2024

Draw indexed, non-instanced primitives.

## Syntax

C++

```
void DrawIndexed(  
    [in] UINT IndexCount,  
    [in] UINT StartIndexLocation,  
    [in] INT BaseVertexLocation  
>;
```

## Parameters

[in] IndexCount

Type: **UINT**

Number of indices to draw.

[in] StartIndexLocation

Type: **UINT**

The location of the first index read by the GPU from the index buffer.

[in] BaseVertexLocation

Type: **INT**

A value added to each index before reading a vertex from the vertex buffer.

## Return value

None

## Remarks

A draw API submits work to the rendering pipeline.

If the sum of both indices is negative, the result of the function call is undefined.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DrawIndexedInstanced method (d3d11.h)

Article 02/22/2024

Draw indexed, instanced primitives.

## Syntax

C++

```
void DrawIndexedInstanced(
    [in] UINT IndexCountPerInstance,
    [in] UINT InstanceCount,
    [in] UINT StartIndexLocation,
    [in] INT BaseVertexLocation,
    [in] UINT StartInstanceLocation
);
```

## Parameters

[in] IndexCountPerInstance

Type: **UINT**

Number of indices read from the index buffer for each instance.

[in] InstanceCount

Type: **UINT**

Number of instances to draw.

[in] StartIndexLocation

Type: **UINT**

The location of the first index read by the GPU from the index buffer.

[in] BaseVertexLocation

Type: **INT**

A value added to each index before reading a vertex from the vertex buffer.

[in] StartInstanceLocation

Type: [UINT](#)

A value added to each index before reading per-instance data from a vertex buffer.

## Return value

None

## Remarks

A draw API submits work to the rendering pipeline.

Instancing may 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.

The second buffer is needed only if the input layout that you use has elements that use [D3D11\\_INPUT\\_PER\\_INSTANCE\\_DATA](#) as the input element classification buffer for per-instance data.

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DrawIndexedInstancedIndirect method (d3d11.h)

Article 02/22/2024

Draw indexed, instanced, GPU-generated primitives.

## Syntax

C++

```
void DrawIndexedInstancedIndirect(
    [in] ID3D11Buffer *pBufferForArgs,
    [in] UINT          AlignedByteOffsetForArgs
);
```

## Parameters

[in] pBufferForArgs

Type: [ID3D11Buffer\\*](#)

A pointer to an [ID3D11Buffer](#), which is a buffer containing the GPU-generated primitives.

[in] AlignedByteOffsetForArgs

Type: [UINT](#)

A DWORD-aligned byte offset in *pBufferForArgs* to the start of the GPU generated primitives.

## Return value

None

## Remarks

When an application creates a buffer that is associated with the [ID3D11Buffer](#) interface that *pBufferForArgs* points to, your application must set the [D3D11\\_RESOURCE\\_MISC\\_DRAWINDIRECT\\_ARGS](#) flag in the *MiscFlags* member of the

[D3D11\\_BUFFER\\_DESC](#) structure that describes the buffer. To create the buffer, your application should call the [ID3D11Device::CreateBuffer](#) method, and pass a pointer to a [D3D11\\_BUFFER\\_DESC](#) in the *pDesc* parameter.

**Windows Phone 8:** This API is supported.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DrawInstanced method (d3d11.h)

Article 02/22/2024

Draw non-indexed, instanced primitives.

## Syntax

C++

```
void DrawInstanced(
    [in] UINT VertexCountPerInstance,
    [in] UINT InstanceCount,
    [in] UINT StartVertexLocation,
    [in] UINT StartInstanceLocation
);
```

## Parameters

[in] VertexCountPerInstance

Type: **UINT**

Number of vertices to draw.

[in] InstanceCount

Type: **UINT**

Number of instances to draw.

[in] StartVertexLocation

Type: **UINT**

Index of the first vertex.

[in] StartInstanceLocation

Type: **UINT**

A value added to each index before reading per-instance data from a vertex buffer.

# Return value

None

## Remarks

A draw API submits work to the rendering pipeline.

Instancing may 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 normally comes from a vertex buffer that is bound to the pipeline. However, you could also provide the vertex data from a shader that has instanced data identified with a system-value semantic (SV\_InstanceID).

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DrawInstancedIndirect method (d3d11.h)

Article 02/22/2024

Draw instanced, GPU-generated primitives.

## Syntax

C++

```
void DrawInstancedIndirect(
    [in] ID3D11Buffer *pBufferForArgs,
    [in] UINT          AlignedByteOffsetForArgs
);
```

## Parameters

[in] pBufferForArgs

Type: [ID3D11Buffer\\*](#)

A pointer to an [ID3D11Buffer](#), which is a buffer containing the GPU generated primitives.

[in] AlignedByteOffsetForArgs

Type: [UINT](#)

Offset in *pBufferForArgs* to the start of the GPU generated primitives.

## Return value

None

## Remarks

When an application creates a buffer that is associated with the [ID3D11Buffer](#) interface that *pBufferForArgs* points to, the application must set the [D3D11\\_RESOURCE\\_MISC\\_DRAWINDIRECT\\_ARGS](#) flag in the [MiscFlags](#) member of the [D3D11\\_BUFFER\\_DESC](#) structure that describes the buffer. To create the buffer, the

application calls the [ID3D11Device::CreateBuffer](#) method and in this call passes a pointer to `D3D11_BUFFER_DESC` in the `pDesc` parameter.

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DSGetConstantBuffers method (d3d11.h)

Article 02/22/2024

Get the constant buffers used by the domain-shader stage.

## Syntax

C++

```
void DSGetConstantBuffers(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - StartSlot).

[out, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*\\*](#)

Array of constant buffer interface pointers (see [ID3D11Buffer](#)) to be returned by the method.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DSGetSamplers method (d3d11.h)

Article 02/22/2024

Get an array of sampler state interfaces from the domain-shader stage.

## Syntax

C++

```
void DSGetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [out, optional] ID3D11SamplerState **ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into a zero-based array to begin getting samplers from (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers to get from a device context. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[out, optional] ppSamplers

Type: [ID3D11SamplerState\\*\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)).

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DSGetShader method (d3d11.h)

Article 02/22/2024

Get the domain shader currently set on the device.

## Syntax

C++

```
void DSGetShader(
    [out]           ID3D11DomainShader  **ppDomainShader,
    [out, optional] ID3D11ClassInstance **ppClassInstances,
    [in, out, optional] UINT          *pNumClassInstances
);
```

## Parameters

[out] ppDomainShader

Type: [ID3D11DomainShader\\*\\*](#)

Address of a pointer to a domain shader (see [ID3D11DomainShader](#)) to be returned by the method.

[out, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*\\*](#)

Pointer to an array of class instance interfaces (see [ID3D11ClassInstance](#)).

[in, out, optional] pNumClassInstances

Type: [UINT\\*](#)

The number of class-instance elements in the array.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DSGetShaderResources method (d3d11.h)

Article 02/22/2024

Get the domain-shader resources.

## Syntax

C++

```
void DSGetShaderResources(
    [in]          UINT           StartSlot,
    [in]          UINT           NumViews,
    [out, optional] ID3D11ShaderResourceView **ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin getting shader resources from (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

The number of resources to get from the device. Up to a maximum of 128 slots are available for shader resources (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[out, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*\\*](#)

Array of [shader resource view](#) interfaces to be returned by the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DSSetConstantBuffers method (d3d11.h)

Article 02/22/2024

Sets the constant buffers used by the domain-shader stage.

## Syntax

C++

```
void DSSetConstantBuffers(  
    [in]          UINT      StartSlot,  
    [in]          UINT      NumBuffers,  
    [in, optional] ID3D11Buffer * const *ppConstantBuffers  
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the zero-based array to begin setting constant buffers to (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to set (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - *StartSlot*).

[in, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*](#)

Array of constant buffers (see [ID3D11Buffer](#)) being given to the device.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The Direct3D 11.1 runtime, which is available starting with Windows 8, can bind a larger number of [ID3D11Buffer](#) resources to the shader than the maximum constant buffer size that is supported by shaders (4096 constants – 432-bit components each). *When you bind such a large buffer, the shader can access only the first 4096 432-bit component constants in the buffer, as if 4096 constants is the full size of the buffer.*

If the application wants the shader to access other parts of the buffer, it must call the [DSSetConstantBuffers1](#) method instead.

**Windows Phone 8:** This API is supported.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DSSetSamplers method (d3d11.h)

Article 02/22/2024

Set an array of sampler states to the domain-shader stage.

## Syntax

C++

```
void DSSetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [in, optional] ID3D11SamplerState * const *ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting samplers to (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers in the array. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[in, optional] ppSamplers

Type: [ID3D11SamplerState\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)). See Remarks.

## Return value

None

## Remarks

Any sampler may be set to **NULL**; this invokes the default state, which is defined to be the following.

```
//Default sampler state:  
D3D11_SAMPLER_DESC SamplerDesc;  
SamplerDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;  
SamplerDesc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.MipLODBias = 0;  
SamplerDesc.MaxAnisotropy = 1;  
SamplerDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;  
SamplerDesc.BorderColor[0] = 1.0f;  
SamplerDesc.BorderColor[1] = 1.0f;  
SamplerDesc.BorderColor[2] = 1.0f;  
SamplerDesc.BorderColor[3] = 1.0f;  
SamplerDesc.MinLOD = -FLT_MAX;  
SamplerDesc.MaxLOD = FLT_MAX;
```

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DSSetShader method (d3d11.h)

Article 02/22/2024

Set a domain shader to the device.

## Syntax

C++

```
void DSSetShader(  
    [in, optional] ID3D11DomainShader  *pDomainShader,  
    [in, optional] ID3D11ClassInstance * const *ppClassInstances,  
                           UINT           NumClassInstances  
) ;
```

## Parameters

[in, optional] pDomainShader

Type: [ID3D11DomainShader\\*](#)

Pointer to a domain shader (see [ID3D11DomainShader](#)). Passing in **NULL** disables the shader for this pipeline stage.

[in, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*](#)

A pointer to an array of class-instance interfaces (see [ID3D11ClassInstance](#)). Each interface used by a shader must have a corresponding class instance or the shader will get disabled. Set ppClassInstances to **NULL** if the shader does not use any interfaces.

NumClassInstances

Type: [UINT](#)

The number of class-instance interfaces in the array.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The maximum number of instances a shader can have is 256.

**Windows Phone 8:** This API is supported.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::DSSetShaderResources method (d3d11.h)

Article 02/22/2024

Bind an array of shader resources to the domain-shader stage.

## Syntax

C++

```
void DSSetShaderResources(
    [in]          UINT             StartSlot,
    [in]          UINT             NumViews,
    [in, optional] ID3D11ShaderResourceView * const *ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting shader resources to (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

Number of shader resources to set. Up to a maximum of 128 slots are available for shader resources(ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[in, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*](#)

Array of [shader resource view](#) interfaces to set to the device.

## Return value

None

## Remarks

If an overlapping resource view is already bound to an output slot, such as a render target, then the method will fill the destination shader resource slot with **NULL**.

For information about creating shader-resource views, see [ID3D11Device::CreateShaderResourceView](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::End method (d3d11.h)

Article 02/22/2024

Mark the end of a series of commands.

## Syntax

C++

```
void End(  
    [in] ID3D11Asynchronous *pAsync  
);
```

## Parameters

[in] pAsync

Type: [ID3D11Asynchronous\\*](#)

A pointer to an [ID3D11Asynchronous](#) interface.

## Return value

None

## Remarks

Use [ID3D11DeviceContext::Begin](#) to mark the beginning of the series of commands.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::ExecuteCommandList method (d3d11.h)

Article 10/13/2021

Queues commands from a command list onto a device.

## Syntax

C++

```
void ExecuteCommandList(  
    [in] ID3D11CommandList *pCommandList,  
    BOOL                 RestoreContextState  
)
```

## Parameters

[in] pCommandList

Type: [ID3D11CommandList\\*](#)

A pointer to an [ID3D11CommandList](#) interface that encapsulates a command list.

RestoreContextState

Type: [BOOL](#)

A Boolean flag that determines whether the target context state is saved prior to and restored after the execution of a command list. Use [TRUE](#) to indicate that the runtime needs to save and restore the state. Use [FALSE](#) to indicate that no state shall be saved or restored, which causes the target context to return to its default state after the command list executes. Applications should typically use [FALSE](#) unless they will restore the state to be nearly equivalent to the state that the runtime would restore if [TRUE](#) were passed. When applications use [FALSE](#), they can avoid unnecessary and inefficient state transitions.

## Return value

None

## Remarks

Use this method to play back a command list that was recorded by a deferred context on any thread.

A call to **ExecuteCommandList** of a command list from a deferred context onto the immediate context is required for the recorded commands to be executed on the graphics processing unit (GPU). A call to **ExecuteCommandList** of a command list from a deferred context onto another deferred context can be used to merge recorded lists. But to run the commands from the merged deferred command list on the GPU, you need to execute them on the immediate context.

This method performs some runtime validation related to queries. Queries that are begun in a device context cannot be manipulated indirectly by executing a command list (that is, Begin or End was invoked against the same query by the deferred context which generated the command list). If such a condition occurs, the **ExecuteCommandList** method does not execute the command list. However, the state of the device context is still maintained, as would be expected ([ID3D11DeviceContext::ClearState](#) is performed, unless the application indicates to preserve the device context state).

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::FinishCommandList method (d3d11.h)

Article 10/13/2021

Create a command list and record graphics commands into it.

## Syntax

C++

```
HRESULT FinishCommandList(
    BOOL           RestoreDeferredContextState,
    [out, optional] ID3D11CommandList **ppCommandList
);
```

## Parameters

RestoreDeferredContextState

Type: **BOOL**

A Boolean flag that determines whether the runtime saves deferred context state before it executes **FinishCommandList** and restores it afterwards. Use **TRUE** to indicate that the runtime needs to save and restore the state. Use **FALSE** to indicate that the runtime will not save or restore any state. In this case, the deferred context will return to its default state after the call to **FinishCommandList** completes. For information about default state, see [ID3D11DeviceContext::ClearState](#). Typically, use **FALSE** unless you restore the state to be nearly equivalent to the state that the runtime would restore if you passed **TRUE**. When you use **FALSE**, you can avoid unnecessary and inefficient state transitions.

**Note** This parameter does not affect the command list that the current call to **FinishCommandList** returns. However, this parameter affects the command list of the next call to **FinishCommandList** on the same deferred context.

[out, optional] ppCommandList

Type: **ID3D11CommandList\*\***

Upon completion of the method, the passed pointer to an [ID3D11CommandList](#) interface pointer is initialized with the recorded command list information. The resulting [ID3D11CommandList](#) object is immutable and can only be used with [ID3D11DeviceContext::ExecuteCommandList](#).

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the following:

- Returns DXGI\_ERROR\_DEVICE\_REMOVED if the video card has been physically removed from the system, or a driver upgrade for the video card has occurred. If this error occurs, you should destroy and recreate the device.
- Returns DXGI\_ERROR\_INVALID\_CALL if [FinishCommandList](#) cannot be called from the current context. See remarks.
- Returns E\_OUTOFMEMORY if the application has exhausted available memory.

## Remarks

Create a command list from a deferred context and record commands into it by calling [FinishCommandList](#). Play back a command list with an immediate context by calling [ID3D11DeviceContext::ExecuteCommandList](#).

Immediate context state is cleared before and after a command list is executed. A command list has no concept of inheritance. Each call to [FinishCommandList](#) will record only the state set since any previous call to [FinishCommandList](#).

For example, the state of a [device context](#) is its render state or pipeline state. To retrieve device context state, an application can call [ID3D11DeviceContext::GetData](#) or [ID3D11DeviceContext::GetPredication](#).

For more information about how to use [FinishCommandList](#), see [How to: Record a Command List](#).

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows

Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?



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

# ID3D11DeviceContext::Flush method (d3d11.h)

Article 06/29/2021

Sends queued-up commands in the command buffer to the graphics processing unit (GPU).

## Syntax

C++

```
void Flush();
```

## Return value

None

## Remarks

Most applications don't need to call this method. If an application calls this method when not necessary, it incurs a performance penalty. Each call to **Flush** incurs a significant amount of overhead.

When Microsoft Direct3D state-setting, present, or draw commands are called by an application, those commands are queued into an internal command buffer. **Flush** sends those commands to the GPU for processing. Typically, the Direct3D runtime sends these commands to the GPU automatically whenever the runtime determines that they need to be sent, such as when the command buffer is full or when an application maps a resource. **Flush** sends the commands manually.

We recommend that you use **Flush** when the CPU waits for an arbitrary amount of time (such as when you call the [Sleep](#) function).

Because **Flush** operates asynchronously, it can return either before or after the GPU finishes executing the queued graphics commands. However, the graphics commands eventually always complete. You can call the [ID3D11Device::CreateQuery](#) method with the [D3D11\\_QUERY\\_EVENT](#) value to create an event query; you can then use that event query in a call to the [ID3D11DeviceContext::GetData](#) method to determine when the GPU is finished processing the graphics commands.

Microsoft Direct3D 11 defers the destruction of objects. Therefore, an application can't rely upon objects immediately being destroyed. By calling **Flush**, you destroy any objects whose destruction was deferred. If an application requires synchronous destruction of an object, we recommend that the application release all its references, call [ID3D11DeviceContext::ClearState](#), and then call **Flush**.

## Deferred Destruction Issues with Flip Presentation Swap Chains

Direct3D 11 defers the destruction of objects like views and resources until it can efficiently destroy them. This deferred destruction can cause problems with flip presentation model swap chains. Flip presentation model swap chains have the [DXGI\\_SWAP\\_EFFECT\\_FLIP\\_SEQUENTIAL](#) flag set. When you create a flip presentation model swap chain, you can associate only one swap chain at a time with an [HWND](#), [IWindow](#), or composition surface. If an application attempts to destroy a flip presentation model swap chain and replace it with another swap chain, the original swap chain is not destroyed when the application immediately frees all of the original swap chain's references.

Most applications typically use the [IDXGISwapChain::ResizeBuffers](#) method for the majority of scenarios where they replace new swap chain buffers for old swap chain buffers. However, if an application must actually destroy an old swap chain and create a new swap chain, the application must force the destruction of all objects that the application freed. To force the destruction, call [ID3D11DeviceContext::ClearState](#) (or otherwise ensure no views are bound to pipeline state), and then call **Flush** on the immediate context. You must force destruction before you call [IDXGIFactory2::CreateSwapChainForHwnd](#), [IDXGIFactory2::CreateSwapChainForCoreWindow](#), or [IDXGIFactory2::CreateSwapChainForComposition](#) again to create a new swap chain.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GenerateMips method (d3d11.h)

Article 07/27/2022

Generates mipmaps for the given shader resource.

## Syntax

C++

```
void GenerateMips(  
    [in] ID3D11ShaderResourceView *pShaderResourceView  
);
```

## Parameters

[in] pShaderResourceView

Type: [ID3D11ShaderResourceView\\*](#)

A pointer to an [ID3D11ShaderResourceView](#) interface that represents the shader resource.

## Return value

None

## Remarks

You can call **GenerateMips** on any shader-resource view to generate the lower mipmap levels for the shader resource. **GenerateMips** uses the largest mipmap level of the view to recursively generate the lower levels of the mip and stops with the smallest level that is specified by the view. If the base resource wasn't created with [D3D11\\_BIND\\_RENDER\\_TARGET](#), [D3D11\\_BIND\\_SHADER\\_RESOURCE](#), and [D3D11\\_RESOURCE\\_MISC\\_GENERATE\\_MIPS](#), the call to **GenerateMips** has no effect.

[Feature levels](#) 9.1, 9.2, and 9.3 can't support automatic generation of mipmaps for 3D (volume) textures.

Video adapters that support [feature level](#) 9.1 and higher support generating mipmaps if you use any of these formats:

```
DXGI_FORMAT_R8G8B8A8_UNORM  
DXGI_FORMAT_R8G8B8A8_UNORM_SRGB  
DXGI_FORMAT_B5G6R5_UNORM  
DXGI_FORMAT_B8G8R8A8_UNORM  
DXGI_FORMAT_B8G8R8A8_UNORM_SRGB  
DXGI_FORMAT_B8G8R8X8_UNORM  
DXGI_FORMAT_B8G8R8X8_UNORM_SRGB
```

Video adapters that support [feature level](#) 9.2 and higher support generating mipmaps if you use any of these formats in addition to any of the formats for feature level 9.1:

```
DXGI_FORMAT_R16G16B16A16_FLOAT  
DXGI_FORMAT_R16G16B16A16_UNORM  
DXGI_FORMAT_R16G16_FLOAT  
DXGI_FORMAT_R16G16_UNORM  
DXGI_FORMAT_R32_FLOAT
```

Video adapters that support [feature level](#) 9.3 and higher support generating mipmaps if you use any of these formats in addition to any of the formats for feature levels 9.1 and 9.2:

```
DXGI_FORMAT_R32G32B32A32_FLOAT  
DXGI_FORMAT_B4G4R4A4 (optional)
```

Video adapters that support [feature level](#) 10 and higher support generating mipmaps if you use any of these formats in addition to any of the formats for feature levels 9.1, 9.2, and 9.3:

```
DXGI_FORMAT_R32G32B32_FLOAT (optional)
```

```
DXGI_FORMAT_R16G16B16A16_SNORM
DXGI_FORMAT_R32G32_FLOAT
DXGI_FORMAT_R10G10B10A2_UNORM
DXGI_FORMAT_R11G11B10_FLOAT
DXGI_FORMAT_R8G8B8A8_SNORM
DXGI_FORMAT_R16G16_SNORM
DXGI_FORMAT_R8G8_UNORM
DXGI_FORMAT_R8G8_SNORM
DXGI_FORMAT_R16_FLOAT
DXGI_FORMAT_R16_UNORM
DXGI_FORMAT_R16_SNORM
DXGI_FORMAT_R8_UNORM
DXGI_FORMAT_R8_SNORM
DXGI_FORMAT_A8_UNORM
DXGI_FORMAT_B5G5R5A1_UNORM (optional)
```

For all other unsupported formats, **GenerateMips** will silently fail.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[DXGI\\_FORMAT](#)

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GetContextFlags method (d3d11.h)

Article02/22/2024

Gets the initialization flags associated with the current deferred context.

## Syntax

C++

```
UINT GetContextFlags();
```

## Return value

None

## Remarks

The `GetContextFlags` method gets the flags that were supplied to the `ContextFlags` parameter of [ID3D11Device::CreateDeferredContext](#); however, the context flag is reserved for future use.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11DeviceContext](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GetData method (d3d11.h)

Article 10/13/2021

Get data from the graphics processing unit (GPU) asynchronously.

## Syntax

C++

```
HRESULT GetData(
    [in]           ID3D11Asynchronous *pAsync,
    [out, optional] void             *pData,
    [in]           UINT            DataSize,
    [in]           UINT            GetDataFlags
);
```

## Parameters

[in] pAsync

Type: [ID3D11Asynchronous\\*](#)

A pointer to an [ID3D11Asynchronous](#) interface for the object about which **GetData** retrieves data.

[out, optional] pData

Type: [void\\*](#)

Address of memory that will receive the data. If **NULL**, **GetData** will be used only to check status. The type of data output depends on the type of asynchronous interface.

[in] DataSize

Type: [UINT](#)

Size of the data to retrieve or 0. Must be 0 when *pData* is **NULL**.

[in] GetDataFlags

Type: [UINT](#)

Optional flags. Can be 0 or any combination of the flags enumerated by [D3D11\\_ASYNC\\_GETDATA\\_FLAG](#).

## Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#). A return value of S\_OK indicates that the data at *pData* is available for the calling application to access. A return value of S\_FALSE indicates that the data is not yet available. If the data is not yet available, the application must call [GetData](#) until the data is available.

## Remarks

Queries in a deferred context are limited to predicated drawing. That is, you cannot call [ID3D11DeviceContext::GetData](#) on a deferred context to get data about a query; you can only call [GetData](#) on the immediate context to get data about a query. For predicated drawing, the results of a predication-type query are used by the GPU and not returned to an application. For more information about predication and predicated drawing, see [D3D11DeviceContext::SetPredication](#).

[GetData](#) retrieves the data that the runtime collected between calls to [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). Certain queries only require a call to [ID3D11DeviceContext::End](#) in which case the data returned by [GetData](#) is accurate up to the last call to [ID3D11DeviceContext::End](#). For information about the queries that only require a call to [ID3D11DeviceContext::End](#) and about the type of data that [GetData](#) retrieves for each query, see [D3D11\\_QUERY](#).

If *DataSize* is 0, [GetData](#) is only used to check status.

An application gathers counter data by calling [ID3D11DeviceContext::Begin](#), issuing some graphics commands, calling [ID3D11DeviceContext::End](#), and then calling [ID3D11DeviceContext::GetData](#) to get data about what happened in between the [Begin](#) and [End](#) calls. For information about performance counter types, see [D3D11\\_COUNTER](#).

## Requirements

Target Platform	Windows
Header	d3d11.h

Library

D3D11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GetPredication method (d3d11.h)

Article 02/22/2024

Get the rendering predicate state.

## Syntax

C++

```
void GetPredication(
    [out, optional] ID3D11Predicate **ppPredicate,
    [out, optional] BOOL           *pPredicateValue
);
```

## Parameters

[out, optional] ppPredicate

Type: [ID3D11Predicate\\*\\*](#)

Address of a pointer to a predicate (see [ID3D11Predicate](#)). Value stored here will be **NULL** upon device creation.

[out, optional] pPredicateValue

Type: [BOOL\\*](#)

Address of a boolean to fill with the predicate comparison value. **FALSE** upon device creation.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call [IUnknown::Release](#) on the returned interfaces when they are no longer needed to avoid memory leaks.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GetResourceMinLOD method (d3d11.h)

Article 02/22/2024

Gets the minimum level-of-detail (LOD).

## Syntax

C++

```
FLOAT GetResourceMinLOD(  
    [in] ID3D11Resource *pResource  
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

A pointer to an [ID3D11Resource](#) which represents the resource.

## Return value

Type: FLOAT

Returns the minimum LOD.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GetType method (d3d11.h)

Article 02/22/2024

Gets the type of [device context](#).

## Syntax

C++

```
D3D11_DEVICE_CONTEXT_TYPE GetType();
```

## Return value

Type: [D3D11\\_DEVICE\\_CONTEXT\\_TYPE](#)

A member of [D3D11\\_DEVICE\\_CONTEXT\\_TYPE](#) that indicates the type of device context.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No



# ID3D11DeviceContext::GSGetConstantBuffers method (d3d11.h)

Article 02/22/2024

Get the constant buffers used by the geometry shader pipeline stage.

## Syntax

C++

```
void GSGetConstantBuffers(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - StartSlot).

[out, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*\\*](#)

Array of constant buffer interface pointers (see [ID3D11Buffer](#)) to be returned by the method.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GSGetSamplers method (d3d11.h)

Article 02/22/2024

Get an array of sampler state interfaces from the geometry shader pipeline stage.

## Syntax

C++

```
void GSGetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [out, optional] ID3D11SamplerState **ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into a zero-based array to begin getting samplers from (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers to get from a device context. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[out, optional] ppSamplers

Type: [ID3D11SamplerState\\*\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)).

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GSGetShader method (d3d11.h)

Article 02/22/2024

Get the geometry shader currently set on the device.

## Syntax

C++

```
void GSGetShader(
    [out]           ID3D11GeometryShader **ppGeometryShader,
    [out, optional] ID3D11ClassInstance **ppClassInstances,
    [in, out, optional] UINT             *pNumClassInstances
);
```

## Parameters

[out] ppGeometryShader

Type: [ID3D11GeometryShader\\*\\*](#)

Address of a pointer to a geometry shader (see [ID3D11GeometryShader](#)) to be returned by the method.

[out, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*\\*](#)

Pointer to an array of class instance interfaces (see [ID3D11ClassInstance](#)).

[in, out, optional] pNumClassInstances

Type: [UINT\\*](#)

The number of class-instance elements in the array.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GSGetShaderResources method (d3d11.h)

Article 02/22/2024

Get the geometry shader resources.

## Syntax

C++

```
void GSGetShaderResources(
    [in]          UINT           StartSlot,
    [in]          UINT           NumViews,
    [out, optional] ID3D11ShaderResourceView **ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin getting shader resources from (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

The number of resources to get from the device. Up to a maximum of 128 slots are available for shader resources (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[out, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*\\*](#)

Array of [shader resource view](#) interfaces to be returned by the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GSSetConstantBuffers method (d3d11.h)

Article 02/22/2024

Sets the constant buffers used by the geometry shader pipeline stage.

## Syntax

C++

```
void GSSetConstantBuffers(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting constant buffers to (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to set (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - *StartSlot*).

[in, optional] ppConstantBuffers

Type: [ID3D11Buffer](#)\*

Array of constant buffers (see [ID3D11Buffer](#)) being given to the device.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

You can't use the [ID3D11ShaderReflectionConstantBuffer](#) interface to get information about what is currently bound to the pipeline in the device context. But you can use [ID3D11ShaderReflectionConstantBuffer](#) to get information from a compiled shader. For example, you can use [ID3D11ShaderReflectionConstantBuffer](#) and [ID3D11ShaderReflectionVariable](#) to determine the slot in which a geometry shader expects a constant buffer. You can then pass this slot number to [GSSetConstantBuffers](#) to set the constant buffer. You can call the [D3D11Reflect](#) function to retrieve the address of a pointer to the [ID3D11ShaderReflection](#) interface and then call [ID3D11ShaderReflection::GetConstantBufferByName](#) to get a pointer to [ID3D11ShaderReflectionConstantBuffer](#).

The Direct3D 11.1 runtime, which is available starting with Windows 8, can bind a larger number of [ID3D11Buffer](#) resources to the shader than the maximum constant buffer size that is supported by shaders (4096 constants – 432-bit components each). *When you bind such a large buffer, the shader can access only the first 4096 432-bit component constants in the buffer, as if 4096 constants is the full size of the buffer.*

If the application wants the shader to access other parts of the buffer, it must call the [GSSetConstantBuffers1](#) method instead.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GSSetSamplers method (d3d11.h)

Article 02/22/2024

Set an array of sampler states to the geometry shader pipeline stage.

## Syntax

C++

```
void GSSetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [in, optional] ID3D11SamplerState * const *ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting samplers to (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers in the array. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[in, optional] ppSamplers

Type: [ID3D11SamplerState\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)). See Remarks.

## Return value

None

## Remarks

Any sampler may be set to **NULL**; this invokes the default state, which is defined to be the following.

```
//Default sampler state:  
D3D11_SAMPLER_DESC SamplerDesc;  
SamplerDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;  
SamplerDesc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.MipLODBias = 0;  
SamplerDesc.MaxAnisotropy = 1;  
SamplerDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;  
SamplerDesc.BorderColor[0] = 1.0f;  
SamplerDesc.BorderColor[1] = 1.0f;  
SamplerDesc.BorderColor[2] = 1.0f;  
SamplerDesc.BorderColor[3] = 1.0f;  
SamplerDesc.MinLOD = -FLT_MAX;  
SamplerDesc.MaxLOD = FLT_MAX;
```

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GSSetShader method (d3d11.h)

Article 02/22/2024

Set a geometry shader to the device.

## Syntax

C++

```
void GSSetShader(  
    [in, optional] ID3D11GeometryShader *pShader,  
    [in, optional] ID3D11ClassInstance  * const *ppClassInstances,  
                           UINT           NumClassInstances  
) ;
```

## Parameters

[in, optional] pShader

Type: [ID3D11GeometryShader\\*](#)

Pointer to a geometry shader (see [ID3D11GeometryShader](#)). Passing in **NULL** disables the shader for this pipeline stage.

[in, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*](#)

A pointer to an array of class-instance interfaces (see [ID3D11ClassInstance](#)). Each interface used by a shader must have a corresponding class instance or the shader will get disabled. Set ppClassInstances to **NULL** if the shader does not use any interfaces.

NumClassInstances

Type: [UINT](#)

The number of class-instance interfaces in the array.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The maximum number of instances a shader can have is 256.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::GSSetShaderResources method (d3d11.h)

Article 02/22/2024

Bind an array of shader resources to the geometry shader stage.

## Syntax

C++

```
void GSSetShaderResources(
    [in]          UINT             StartSlot,
    [in]          UINT             NumViews,
    [in, optional] ID3D11ShaderResourceView * const *ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting shader resources to (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

Number of shader resources to set. Up to a maximum of 128 slots are available for shader resources(ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[in, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*](#)

Array of [shader resource view](#) interfaces to set to the device.

## Return value

None

## Remarks

If an overlapping resource view is already bound to an output slot, such as a render target, then the method will fill the destination shader resource slot with **NULL**.

For information about creating shader-resource views, see [ID3D11Device::CreateShaderResourceView](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::HSGetConstantBuffers method (d3d11.h)

Article 02/22/2024

Get the constant buffers used by the [hull-shader stage](#).

## Syntax

C++

```
void HSGetConstantBuffers(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - StartSlot).

[out, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*\\*](#)

Array of constant buffer interface pointers (see [ID3D11Buffer](#)) to be returned by the method.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::HSGetSamplers method (d3d11.h)

Article 02/22/2024

Get an array of sampler state interfaces from the [hull-shader stage](#).

## Syntax

C++

```
void HSGetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [out, optional] ID3D11SamplerState **ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into a zero-based array to begin getting samplers from (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers to get from a device context. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[out, optional] ppSamplers

Type: [ID3D11SamplerState\\*\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)).

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::HSGetShader method (d3d11.h)

Article 02/22/2024

Get the hull shader currently set on the device.

## Syntax

C++

```
void HSGetShader(
    [out]           ID3D11HullShader    **ppHullShader,
    [out, optional] ID3D11ClassInstance **ppClassInstances,
    [in, out, optional] UINT            *pNumClassInstances
);
```

## Parameters

[out] ppHullShader

Type: [ID3D11HullShader\\*\\*](#)

Address of a pointer to a hull shader (see [ID3D11HullShader](#)) to be returned by the method.

[out, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*\\*](#)

Pointer to an array of class instance interfaces (see [ID3D11ClassInstance](#)).

[in, out, optional] pNumClassInstances

Type: [UINT\\*](#)

The number of class-instance elements in the array.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::HSGetShaderResources method (d3d11.h)

Article 02/22/2024

Get the hull-shader resources.

## Syntax

C++

```
void HSGetShaderResources(
    [in]          UINT           StartSlot,
    [in]          UINT           NumViews,
    [out, optional] ID3D11ShaderResourceView **ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin getting shader resources from (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

The number of resources to get from the device. Up to a maximum of 128 slots are available for shader resources (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[out, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*\\*](#)

Array of [shader resource view](#) interfaces to be returned by the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::HSSetConstantBuffers method (d3d11.h)

Article 02/22/2024

Set the constant buffers used by the [hull-shader stage](#).

## Syntax

C++

```
void HSSetConstantBuffers(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting constant buffers to (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to set (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - *StartSlot*).

[in, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*](#)

Array of constant buffers (see [ID3D11Buffer](#)) being given to the device.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The Direct3D 11.1 runtime, which is available starting with Windows 8, can bind a larger number of [ID3D11Buffer](#) resources to the shader than the maximum constant buffer size that is supported by shaders (4096 constants – 432-bit components each). *When you bind such a large buffer, the shader can access only the first 4096 432-bit component constants in the buffer, as if 4096 constants is the full size of the buffer.*

If the application wants the shader to access other parts of the buffer, it must call the [HSSetConstantBuffers1](#) method instead.

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::HSSetSamplers method (d3d11.h)

Article 02/22/2024

Set an array of sampler states to the [hull-shader stage](#).

## Syntax

C++

```
void HSSetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [in, optional] ID3D11SamplerState * const *ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the zero-based array to begin setting samplers to (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers in the array. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[in, optional] ppSamplers

Type: [ID3D11SamplerState\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)). See Remarks.

## Return value

None

## Remarks

Any sampler may be set to **NULL**; this invokes the default state, which is defined to be the following.

```
//Default sampler state:  
D3D11_SAMPLER_DESC SamplerDesc;  
SamplerDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;  
SamplerDesc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.MipLODBias = 0;  
SamplerDesc.MaxAnisotropy = 1;  
SamplerDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;  
SamplerDesc.BorderColor[0] = 1.0f;  
SamplerDesc.BorderColor[1] = 1.0f;  
SamplerDesc.BorderColor[2] = 1.0f;  
SamplerDesc.BorderColor[3] = 1.0f;  
SamplerDesc.MinLOD = -FLT_MAX;  
SamplerDesc.MaxLOD = FLT_MAX;
```

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::HSSetShader method (d3d11.h)

Article 10/13/2021

Set a hull shader to the device.

## Syntax

C++

```
void HSSetShader(  
    [in, optional] ID3D11HullShader     *pHullShader,  
    [in, optional] ID3D11ClassInstance * const *ppClassInstances,  
                           UINT           NumClassInstances  
) ;
```

## Parameters

[in, optional] pHullShader

Type: [ID3D11HullShader\\*](#)

Pointer to a hull shader (see [ID3D11HullShader](#)). Passing in **NULL** disables the shader for this pipeline stage.

[in, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*](#)

A pointer to an array of class-instance interfaces (see [ID3D11ClassInstance](#)). Each interface used by a shader must have a corresponding class instance or the shader will get disabled. Set ppClassInstances to **NULL** if the shader does not use any interfaces.

NumClassInstances

Type: [UINT](#)

The number of class-instance interfaces in the array.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The maximum number of instances a shader can have is 256.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::HSSetShaderResources method (d3d11.h)

Article 02/22/2024

Bind an array of shader resources to the [hull-shader stage](#).

## Syntax

C++

```
void HSSetShaderResources(
    [in]          UINT             StartSlot,
    [in]          UINT             NumViews,
    [in, optional] ID3D11ShaderResourceView * const *ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting shader resources to (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

Number of shader resources to set. Up to a maximum of 128 slots are available for shader resources(ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[in, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*](#)

Array of [shader resource view](#) interfaces to set to the device.

## Return value

None

## Remarks

If an overlapping resource view is already bound to an output slot, such as a render target, then the method will fill the destination shader resource slot with **NULL**.

For information about creating shader-resource views, see [ID3D11Device::CreateShaderResourceView](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::IAGetIndexBuffer method (d3d11.h)

Article 02/22/2024

Get a pointer to the index buffer that is bound to the input-assembler stage.

## Syntax

C++

```
void IAGetIndexBuffer(
    [out, optional] ID3D11Buffer **pIndexBuffer,
    [out, optional] DXGI_FORMAT *Format,
    [out, optional] UINT *Offset
);
```

## Parameters

[out, optional] pIndexBuffer

Type: [ID3D11Buffer\\*\\*](#)

A pointer to an index buffer returned by the method (see [ID3D11Buffer](#)).

[out, optional] Format

Type: [DXGI\\_FORMAT\\*](#)

Specifies format of the data in the index buffer (see [DXGI\\_FORMAT](#)). These formats provide the size and type of the data in the buffer. The only formats allowed for index buffer data are 16-bit (DXGI\_FORMAT\_R16\_UINT) and 32-bit (DXGI\_FORMAT\_R32\_UINT) integers.

[out, optional] Offset

Type: [UINT\\*](#)

Offset (in bytes) from the start of the index buffer, to the first index to use.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::IAGetInputLayout method (d3d11.h)

Article 07/27/2022

Get a pointer to the input-layout object that is bound to the input-assembler stage.

## Syntax

C++

```
void IAGetInputLayout(  
    [out] ID3D11InputLayout **ppInputLayout  
) ;
```

## Parameters

[out] ppInputLayout

Type: [ID3D11InputLayout\\*\\*](#)

A pointer to the input-layout object (see [ID3D11InputLayout](#)), which describes the input buffers that will be read by the IA stage.

## Return value

None

## Remarks

For information about creating an input-layout object, see [Creating the Input-Layout Object](#).

Any returned interfaces will have their reference count incremented by one. Applications should call [IUnknown::Release](#) on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::IAGetPrimitiveTopology method (d3d11.h)

Article 02/22/2024

Get information about the primitive type, and data order that describes input data for the input assembler stage.

## Syntax

C++

```
void IAGetPrimitiveTopology(  
    [out] D3D11_PRIMITIVE_TOPOLOGY *pTopology  
) ;
```

## Parameters

[out] pTopology

Type: [D3D11\\_PRIMITIVE\\_TOPOLOGY\\*](#)

A pointer to the type of primitive, and ordering of the primitive data (see [D3D11\\_PRIMITIVE\\_TOPOLOGY](#)).

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::IAGetVertexBuffers method (d3d11.h)

Article 02/22/2024

Get the vertex buffers bound to the input-assembler stage.

## Syntax

C++

```
void IAGetVertexBuffers(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppVertexBuffers,
    [out, optional] UINT      *pStrides,
    [out, optional] UINT      *pOffsets
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

The input slot of the first vertex buffer to get. The first vertex buffer is explicitly bound to the start slot; this causes each additional vertex buffer in the array to be implicitly bound to each subsequent input slot. The maximum of 16 or 32 input slots (ranges from 0 to D3D11\_IA\_VERTEX\_INPUT\_RESOURCE\_SLOT\_COUNT - 1) are available; the [maximum number of input slots depends on the feature level](#).

[in] NumBuffers

Type: [UINT](#)

The number of vertex buffers to get starting at the offset. The number of buffers (plus the starting slot) cannot exceed the total number of IA-stage input slots.

[out, optional] ppVertexBuffers

Type: [ID3D11Buffer\\*\\*](#)

A pointer to an array of vertex buffers returned by the method (see [ID3D11Buffer](#)).

[out, optional] pStrides

Type: **UINT\***

Pointer to an array of stride values returned by the method; one stride value for each buffer in the vertex-buffer array. Each stride value is the size (in bytes) of the elements that are to be used from that vertex buffer.

[out, optional] pOffsets

Type: **UINT\***

Pointer to an array of offset values returned by the method; one offset value for each buffer in the vertex-buffer array. Each offset is the number of bytes between the first element of a vertex buffer and the first element that will be used.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::IASetIndexBuffer method (d3d11.h)

Article 07/27/2022

Bind an index buffer to the input-assembler stage.

## Syntax

C++

```
void IASetIndexBuffer(
    [in, optional] ID3D11Buffer *pIndexBuffer,
    [in]           DXGI_FORMAT Format,
    [in]           UINT       Offset
);
```

## Parameters

[in, optional] pIndexBuffer

Type: [ID3D11Buffer\\*](#)

A pointer to an [ID3D11Buffer](#) object, that contains indices. The index buffer must have been created with the [D3D11\\_BIND\\_INDEX\\_BUFFER](#) flag.

[in] Format

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#) that specifies the format of the data in the index buffer. The only formats allowed for index buffer data are 16-bit ([DXGI\\_FORMAT\\_R16\\_UINT](#)) and 32-bit ([DXGI\\_FORMAT\\_R32\\_UINT](#)) integers.

[in] Offset

Type: [UINT](#)

Offset (in bytes) from the start of the index buffer to the first index to use.

## Return value

None

## Remarks

For information about creating index buffers, see [How to: Create an Index Buffer](#).

Calling this method using a buffer that is currently bound for writing (i.e. bound to the stream output pipeline stage) will effectively bind **NULL** instead because a buffer cannot be bound as both an input and an output at the same time.

The debug layer will generate a warning whenever a resource is prevented from being bound simultaneously as an input and an output, but this will not prevent invalid data from being used by the runtime.

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::IASetInputLayout method (d3d11.h)

Article 07/27/2022

Bind an input-layout object to the input-assembler stage.

## Syntax

C++

```
void IASetInputLayout(  
    [in, optional] ID3D11InputLayout *pInputLayout  
);
```

## Parameters

[in, optional] pInputLayout

Type: [ID3D11InputLayout\\*](#)

A pointer to the input-layout object (see [ID3D11InputLayout](#)), which describes the input buffers that will be read by the IA stage.

## Return value

None

## Remarks

Input-layout objects describe how vertex buffer data is streamed into the IA pipeline stage. To create an input-layout object, call [ID3D11Device::CreateInputLayout](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::IASetPrimitiveTopology method (d3d11.h)

Article 02/22/2024

Bind information about the primitive type, and data order that describes input data for the input assembler stage.

## Syntax

C++

```
void IASetPrimitiveTopology(  
    [in] D3D11_PRIMITIVE_TOPOLOGY Topology  
)
```

## Parameters

[in] Topology

Type: [D3D11\\_PRIMITIVE\\_TOPOLOGY](#)

The type of primitive and ordering of the primitive data (see [D3D11\\_PRIMITIVE\\_TOPOLOGY](#)).

## Return value

None

## Remarks

Windows Phone 8: This API is supported.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::IASetVertexBuffers method (d3d11.h)

Article 02/22/2024

Bind an array of vertex buffers to the [input-assembler stage](#).

## Syntax

C++

```
void IASetVertexBuffers(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppVertexBuffers,
    [in, optional] const UINT   *pStrides,
    [in, optional] const UINT   *pOffsets
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

The first input slot for binding. The first vertex buffer is explicitly bound to the start slot; this causes each additional vertex buffer in the array to be implicitly bound to each subsequent input slot. The maximum of 16 or 32 input slots (ranges from 0 to D3D11\_IA\_VERTEX\_INPUT\_RESOURCE\_SLOT\_COUNT - 1) are available; the [maximum number of input slots depends on the feature level](#).

[in] NumBuffers

Type: [UINT](#)

The number of vertex buffers in the array. The number of buffers (plus the starting slot) can't exceed the total number of IA-stage input slots (ranges from 0 to D3D11\_IA\_VERTEX\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[in, optional] ppVertexBuffers

Type: [ID3D11Buffer\\*](#)

A pointer to an array of vertex buffers (see [ID3D11Buffer](#)). The vertex buffers must have been created with the [D3D11\\_BIND\\_VERTEX\\_BUFFER](#) flag.

[in, optional] pStrides

Type: **const UINT\***

Pointer to an array of stride values; one stride value for each buffer in the vertex-buffer array. Each stride is the size (in bytes) of the elements that are to be used from that vertex buffer.

[in, optional] pOffsets

Type: **const UINT\***

Pointer to an array of offset values; one offset value for each buffer in the vertex-buffer array. Each offset is the number of bytes between the first element of a vertex buffer and the first element that will be used.

## Return value

None

## Remarks

For info about creating vertex buffers, see [How to: Create a Vertex Buffer](#).

Calling this method using a buffer that is currently bound for writing (that is, bound to the stream output pipeline stage) will effectively bind **NULL** instead because a buffer can't be bound as both an input and an output at the same time.

The debug layer will generate a warning whenever a resource is prevented from being bound simultaneously as an input and an output, but this will not prevent invalid data from being used by the runtime.

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

**Windows Phone 8:** This API is supported.

## Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::Map method (d3d11.h)

Article 10/13/2021

Gets a pointer to the data contained in a [subresource](#), and denies the GPU access to that subresource.

## Syntax

C++

```
HRESULT Map(
    [in]          ID3D11Resource      *pResource,
    [in]          UINT              Subresource,
    [in]          D3D11_MAP        MapType,
    [in]          UINT              MapFlags,
    [out, optional] D3D11_MAPPED_SUBRESOURCE *pMappedResource
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource](#)\*

A pointer to a [ID3D11Resource](#) interface.

[in] Subresource

Type: [UINT](#)

Index number of the [subresource](#).

[in] MapType

Type: [D3D11\\_MAP](#)

A [D3D11\\_MAP](#)-typed value that specifies the CPU's read and write permissions for a resource.

[in] MapFlags

Type: [UINT](#)

Flag that specifies what the CPU does when the GPU is busy. This flag is optional.

[out, optional] pMappedResource

Type: [D3D11\\_MAPPED\\_SUBRESOURCE\\*](#)

A pointer to the [D3D11\\_MAPPED\\_SUBRESOURCE](#) structure for the mapped subresource. See the Remarks section regarding NULL pointers.

## Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

This method also returns [DXGI\\_ERROR\\_WAS\\_STILL\\_DRAWING](#) if *MapFlags* specifies [D3D11\\_MAP\\_FLAG\\_DO\\_NOT\\_WAIT](#) and the GPU is not yet finished with the resource.

This method also returns [DXGI\\_ERROR\\_DEVICE\\_REMOVED](#) if *MapType* allows any CPU read access and the video card has been removed.

For more information about these error codes, see [DXGI\\_ERROR](#).

## Remarks

If you call **Map** on a deferred context, you can only pass [D3D11\\_MAP\\_WRITE\\_DISCARD](#), [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#), or both to the *MapType* parameter. Other [D3D11\\_MAP](#)-typed values are not supported for a deferred context.

**Note** The Direct3D 11.1 runtime, which is available starting with Windows 8, enables mapping dynamic constant buffers and shader resource views (SRVs) of dynamic buffers with [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#). The Direct3D 11 and earlier runtimes limited mapping to vertex or index buffers. To determine if a Direct3D device supports these features, call [ID3D11Device::CheckFeatureSupport](#) with [D3D11\\_FEATURE\\_D3D11\\_OPTIONS](#). [CheckFeatureSupport](#) fills members of a [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) structure with the device's features. The relevant members here are [MapNoOverwriteOnDynamicConstantBuffer](#) and [MapNoOverwriteOnDynamicBufferSRV](#).

For info about how to use **Map**, see [How to: Use dynamic resources](#).

## NULL pointers for pMappedResource

The *pMappedResource* parameter may be NULL when a texture is provided that was created with [D3D11\\_USAGE\\_DEFAULT](#), and the API is called on an immediate context. This allows a default texture to be mapped, even if it was created using [D3D11\\_TEXTURE\\_LAYOUT\\_UNDEFINED](#). Following this API call, the texture may be accessed using [ID3D11DeviceContext3::WriteToSubresource](#) and/or [ID3D11DeviceContext3::ReadFromSubresource](#).

## Don't read from a subresource mapped for writing

When you pass [D3D11\\_MAP\\_WRITE](#), [D3D11\\_MAP\\_WRITE\\_DISCARD](#), or [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#) to the *MapType* parameter, you must ensure that your app does not read the subresource data to which the **pData** member of [D3D11\\_MAPPED\\_SUBRESOURCE](#) points because doing so can cause a significant performance penalty. 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.

### Note

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.

Windows Phone 8: This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::OMGetBlendState method (d3d11.h)

Article 02/22/2024

Get the blend state of the output-merger stage.

## Syntax

C++

```
void OMGetBlendState(
    [out, optional] ID3D11BlendState **ppBlendState,
    [out, optional] FLOAT [4] BlendFactor,
    [out, optional] UINT *pSampleMask
);
```

## Parameters

[out, optional] ppBlendState

Type: [ID3D11BlendState\\*\\*](#)

Address of a pointer to a blend-state interface (see [ID3D11BlendState](#)).

[out, optional] BlendFactor

Type: [FLOAT\[4\]](#)

Array of blend factors, one for each RGBA component.

[out, optional] pSampleMask

Type: [UINT\\*](#)

Pointer to a [sample mask](#).

## Return value

None

## Remarks

The reference count of the returned interface will be incremented by one when the blend state is retrieved. Applications must release returned pointer(s) when they are no longer needed, or else there will be a memory leak.

**Windows Phone 8:** This API is supported.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::OMGetDepthStencilState method (d3d11.h)

Article 02/22/2024

Gets the depth-stencil state of the output-merger stage.

## Syntax

C++

```
void OMGetDepthStencilState(
    [out, optional] ID3D11DepthStencilState **ppDepthStencilState,
    [out, optional] UINT                 *pStencilRef
);
```

## Parameters

[out, optional] ppDepthStencilState

Type: [ID3D11DepthStencilState\\*\\*](#)

Address of a pointer to a depth-stencil state interface (see [ID3D11DepthStencilState](#)) to be filled with information from the device.

[out, optional] pStencilRef

Type: [UINT\\*](#)

Pointer to the stencil reference value used in the depth-stencil test.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call [IUnknown::Release](#) on the returned interfaces when they are no longer needed to avoid memory leaks.

Windows Phone 8: This API is supported.

# Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::OMGetRenderTargets method (d3d11.h)

Article 07/27/2022

Get pointers to the resources bound to the output-merger stage.

## Syntax

C++

```
void OMGetRenderTargets(
    [in]          UINT             NumViews,
    [out, optional] ID3D11RenderTargetView **ppRenderTargetViews,
    [out, optional] ID3D11DepthStencilView **ppDepthStencilView
);
```

## Parameters

[in] NumViews

Type: [UINT](#)

Number of render targets to retrieve.

[out, optional] ppRenderTargetViews

Type: [ID3D11RenderTargetView\\*\\*](#)

Pointer to an array of [ID3D11RenderTargetViews](#) which represent render target views. Specify **NULL** for this parameter when retrieval of a render target is not needed.

[out, optional] ppDepthStencilView

Type: [ID3D11DepthStencilView\\*\\*](#)

Pointer to a [ID3D11DepthStencilView](#), which represents a depth-stencil view. Specify **NULL** for this parameter when retrieval of the depth-stencil view is not needed.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::OMGetRenderTargetsAndUnorderedAccessViews method (d3d11.h)

Article 02/22/2024

Get pointers to the resources bound to the output-merger stage.

## Syntax

C++

```
void OMGetRenderTargetsAndUnorderedAccessViews(
    [in]          UINT           NumRTVs,
    [out, optional] ID3D11RenderTargetView **ppRenderTargetViews,
    [out, optional] ID3D11DepthStencilView **ppDepthStencilView,
    [in]          UINT           UAVStartSlot,
    [in]          UINT           NumUAVs,
    [out, optional] ID3D11UnorderedAccessView **ppUnorderedAccessViews
);
```

## Parameters

[in] NumRTVs

Type: [UINT](#)

The number of render-target views to retrieve.

[out, optional] ppRenderTargetViews

Type: [ID3D11RenderTargetView\\*\\*](#)

Pointer to an array of [ID3D11RenderTargetViews](#), which represent render-target views. Specify **NULL** for this parameter when retrieval of render-target views is not required.

[out, optional] ppDepthStencilView

Type: [ID3D11DepthStencilView\\*\\*](#)

Pointer to a [ID3D11DepthStencilView](#), which represents a depth-stencil view. Specify **NULL** for this parameter when retrieval of the depth-stencil view is not required.

[in] `UAVStartSlot`

Type: **UINT**

Index into a zero-based array to begin retrieving unordered-access views (ranges from 0 to `D3D11_PS_CS_UAV_REGISTER_COUNT` - 1). For pixel shaders `UAVStartSlot` should be equal to the number of render-target views that are bound.

[in] `NumUVAs`

Type: **UINT**

Number of unordered-access views to return in `ppUnorderedAccessViews`. This number ranges from 0 to `D3D11_PS_CS_UAV_REGISTER_COUNT` - `UAVStartSlot`.

[out, optional] `ppUnorderedAccessViews`

Type: **ID3D11UnorderedAccessView\*\***

Pointer to an array of `ID3D11UnorderedAccessViews`, which represent unordered-access views that are retrieved. Specify `NULL` for this parameter when retrieval of unordered-access views is not required.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

**Windows Phone 8:** This API is supported.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::OMSetBlendState method (d3d11.h)

Article 07/27/2022

Set the blend state of the output-merger stage.

## Syntax

C++

```
void OMSetBlendState(
    [in, optional] ID3D11BlendState *pBlendState,
    [in, optional] const FLOAT [4] BlendFactor,
    [in]           UINT          SampleMask
);
```

## Parameters

[in, optional] pBlendState

Type: [ID3D11BlendState\\*](#)

Pointer to a blend-state interface (see [ID3D11BlendState](#)). Pass **NULL** for a default blend state. For more info about default blend state, see Remarks.

[in, optional] BlendFactor

Type: [const FLOAT\[4\]](#)

Array of blend factors, one for each RGBA component. The blend factors modulate values for the pixel shader, render target, or both. If you created the blend-state object with [D3D11\\_BLEND\\_BLEND\\_FACTOR](#) or [D3D11\\_BLEND\\_INV\\_BLEND\\_FACTOR](#), the blending stage uses the non-NULL array of blend factors. If you didn't create the blend-state object with [D3D11\\_BLEND\\_BLEND\\_FACTOR](#) or [D3D11\\_BLEND\\_INV\\_BLEND\\_FACTOR](#), the blending stage does not use the non-NULL array of blend factors; the runtime stores the blend factors, and you can later call [ID3D11DeviceContext::OMGetBlendState](#) to retrieve the blend factors. If you pass **NULL**, the runtime uses or stores a blend factor equal to { 1, 1, 1, 1 }.

[in] SampleMask

Type: **UINT**

32-bit sample coverage. The default value is 0xffffffff. See remarks.

## Return value

None

## Remarks

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 [blend option](#) controls the data source that the blending stage uses to modulate values for the pixel shader, render target, or both. The [blend operation](#) controls how the blending stage mathematically combines these modulated values.

To create a blend-state interface, call [ID3D11Device::CreateBlendState](#).

Passing in **NULL** for the blend-state interface indicates to the runtime to set a default blending state. The following table indicates the default blending parameters.

State	Default Value
AlphaToCoverageEnable	FALSE
IndependentBlendEnable	FALSE
RenderTarget[0].BlendEnable	FALSE
RenderTarget[0].SrcBlend	D3D11_BLEND_ONE
RenderTarget[0].DestBlend	D3D11_BLEND_ZERO
RenderTarget[0].BlendOp	D3D11_BLEND_OP_ADD
RenderTarget[0].SrcBlendAlpha	D3D11_BLEND_ONE
RenderTarget[0].DestBlendAlpha	D3D11_BLEND_ZERO
RenderTarget[0].BlendOpAlpha	D3D11_BLEND_OP_ADD
RenderTarget[0].RenderTargetWriteMask	D3D11_COLOR_WRITE_ENABLE_ALL

A sample mask determines which samples get updated in all the active render targets. The mapping of bits in a sample mask to samples in a multisample render target is the responsibility of an individual application. A sample mask is always applied; it is independent of whether multisampling is enabled, and does not depend on whether an application uses multisample render targets.

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::OMSetDepthStencilState method (d3d11.h)

Article 02/22/2024

Sets the depth-stencil state of the output-merger stage.

## Syntax

C++

```
void OMSetDepthStencilState(
    [in, optional] ID3D11DepthStencilState *pDepthStencilState,
    [in]           UINT                 StencilRef
);
```

## Parameters

[in, optional] pDepthStencilState

Type: [ID3D11DepthStencilState\\*](#)

Pointer to a depth-stencil state interface (see [ID3D11DepthStencilState](#)) to bind to the device. Set this to **NULL** to use the default state listed in [D3D11\\_DEPTH\\_STENCIL\\_DESC](#).

[in] StencilRef

Type: [UINT](#)

Reference value to perform against when doing a depth-stencil test. See remarks.

## Return value

None

## Remarks

To create a depth-stencil state interface, call [ID3D11Device::CreateDepthStencilState](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::OMSetRenderTargets method (d3d11.h)

Article 11/23/2021

Bind one or more render targets atomically and the depth-stencil buffer to the [output-merger stage](#).

## Syntax

C++

```
void OMSetRenderTargets(
    [in]          UINT           NumViews,
    [in, optional] ID3D11RenderTargetView * const *ppRenderTargetViews,
    [in, optional] ID3D11DepthStencilView *pDepthStencilView
);
```

## Parameters

[in] NumViews

Type: [UINT](#)

Number of render targets to bind (ranges between 0 and [D3D11\\_SIMULTANEOUS\\_RENDER\\_TARGET\\_COUNT](#)). If this parameter is nonzero, the number of entries in the array to which *ppRenderTargetViews* points must equal the number in this parameter.

[in, optional] ppRenderTargetViews

Type: [ID3D11RenderTargetView\\*](#)

Pointer to an array of [ID3D11RenderTargetView](#) that represent the render targets to bind to the device. If this parameter is **NULL** and *NumViews* is 0, no render targets are bound.

[in, optional] pDepthStencilView

Type: [ID3D11DepthStencilView\\*](#)

Pointer to a [ID3D11DepthStencilView](#) that represents the depth-stencil view to bind to the device. If this parameter is **NULL**, the depth-stencil view is not bound.

# Return value

None

## Remarks

The maximum number of active render targets a device can have active at any given time is set by a #define in D3D11.h called

**D3D11\_SIMULTANEOUS\_RENDER\_TARGET\_COUNT**. It is invalid to try to set the same subresource to multiple render target slots. Any render targets not defined by this call are set to **NULL**.

If any subresources are also currently bound for reading in a different stage or writing (perhaps in a different part of the pipeline), those bind points will be set to **NULL**, in order to prevent the same subresource from being read and written simultaneously in a single rendering operation.

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

If the render-target views were created from an array resource type, all of the render-target views must have the same array size.

This restriction also applies to the depth-stencil view, its array size must match that of the render-target views being bound.

The pixel shader must be able to simultaneously render to at least eight separate render targets. All of these render targets must access the same type of resource: [Buffer](#), [Texture1D](#), [Texture1DArray](#), [Texture2D](#), [Texture2DArray](#), [Texture3D](#), or [TextureCube](#). All render targets must have the same size in all dimensions (width and height, and depth for 3D or array size for \*Array types). If render targets use multisample anti-aliasing, all bound render targets and depth buffer must be the same form of multisample resource (that is, the sample counts must be the same). Each render target can have a different data format. These render target formats are not required to have identical bit-per-element counts.

Any combination of the eight slots for render targets can have a render target set or not set.

The same resource view cannot be bound to multiple render target slots simultaneously. However, you can set multiple non-overlapping resource views of a single resource as simultaneous multiple render targets.

Note that unlike some other resource methods in Direct3D, all currently bound render targets will be unbound by calling `OMSetRenderTargets(0, nullptr, nullptr);`.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::OMSetRenderTargetsAndUnorderedAccessViews method (d3d11.h)

Article 10/13/2021

Binds resources to the output-merger stage.

## Syntax

C++

```
void OMSetRenderTargetsAndUnorderedAccessViews(
    [in]          UINT           NumRTVs,
    [in, optional] ID3D11RenderTargetView * const *ppRenderTargetViews,
    [in, optional] ID3D11DepthStencilView   *pDepthStencilView,
    [in]          UINT           UAVStartSlot,
    [in]          UINT           NumUAVs,
    [in, optional] ID3D11UnorderedAccessView * const *ppUnorderedAccessViews,
    [in, optional] const UINT        *pUAVInitialCounts
);
```

## Parameters

[in] NumRTVs

Type: [UINT](#)

Number of render targets to bind (ranges between 0 and [D3D11\\_SIMULTANEOUS\\_RENDER\\_TARGET\\_COUNT](#)). If this parameter is nonzero, the number of entries in the array to which *ppRenderTargetViews* points must equal the number in this parameter. If you set *NumRTVs* to [D3D11\\_KEEP\\_RENDER\\_TARGETS\\_AND\\_DEPTH\\_STENCIL](#) (0xffffffff), this method does not modify the currently bound render-target views (RTVs) and also does not modify depth-stencil view (DSV).

[in, optional] ppRenderTargetViews

Type: [ID3D11RenderTargetView\\*](#)

Pointer to an array of [ID3D11RenderTargetViews](#) that represent the render targets to bind to the device. If this parameter is [NULL](#) and *NumRTVs* is 0, no render targets are

bound.

[in, optional] pDepthStencilView

Type: [ID3D11DepthStencilView\\*](#)

Pointer to a [ID3D11DepthStencilView](#) that represents the depth-stencil view to bind to the device. If this parameter is **NULL**, the depth-stencil view is not bound.

[in] UAVStartSlot

Type: [UINT](#)

Index into a zero-based array to begin setting unordered-access views (ranges from 0 to D3D11\_PS\_CS\_UAV\_REGISTER\_COUNT - 1).

For the Direct3D 11.1 runtime, which is available starting with Windows 8, this value can range from 0 to D3D11\_1\_UAV\_SLOT\_COUNT - 1. D3D11\_1\_UAV\_SLOT\_COUNT is defined as 64.

For pixel shaders, *UAVStartSlot* should be equal to the number of render-target views being bound.

[in] NumUAVs

Type: [UINT](#)

Number of unordered-access views (UAVs) in *ppUnorderedAccessViews*. If you set *NumUAVs* to D3D11\_KEEP\_UNORDERED\_ACCESS\_VIEWS (0xffffffff), this method does not modify the currently bound unordered-access views.

For the Direct3D 11.1 runtime, which is available starting with Windows 8, this value can range from 0 to D3D11\_1\_UAV\_SLOT\_COUNT - *UAVStartSlot*.

[in, optional] ppUnorderedAccessViews

Type: [ID3D11UnorderedAccessView\\*](#)

Pointer to an array of [ID3D11UnorderedAccessViews](#) that represent the unordered-access views to bind to the device. If this parameter is **NULL** and *NumUAVs* is 0, no unordered-access views are bound.

[in, optional] pUAVInitialCounts

Type: [const UINT\\*](#)

An array of append and consume buffer offsets. A value of -1 indicates to keep the current offset. Any other values set the hidden counter for that appendable and consumable UAV. *pUAVInitialCounts* is relevant only for UAVs that were created with either [D3D11\\_BUFFER\\_UAV\\_FLAG\\_APPEND](#) or [D3D11\\_BUFFER\\_UAV\\_FLAG\\_COUNTER](#) specified when the UAV was created; otherwise, the argument is ignored.

## Return value

None

## Remarks

For pixel shaders, the render targets and unordered-access views share the same resource slots when being written out. This means that UAVs must be given an offset so that they are placed in the slots after the render target views that are being bound.

**Note** RTVs, DSV, and UAVs cannot be set independently; they all need to be set at the same time.

Two RTVs conflict if they share a subresource (and therefore share the same resource).

Two UAVs conflict if they share a subresource (and therefore share the same resource).

An RTV conflicts with a UAV if they share a subresource or share a bind point.

**OMSetRenderTargetsAndUnorderedAccessViews** operates properly in the following situations:

1. *NumRTVs* != [D3D11\\_KEEP\\_RENDER\\_TARGETS\\_AND\\_DEPTH\\_STENCIL](#) and *NumUAVs* != [D3D11\\_KEEP\\_UNORDERED\\_ACCESS\\_VIEWS](#)

The following conditions must be true for

**OMSetRenderTargetsAndUnorderedAccessViews** to succeed and for the runtime to pass the bind information to the driver:

- *NumRTVs* <= 8
- *UAVStartSlot* >= *NumRTVs*
- *UAVStartSlot* + *NumUAVs* <= 8
- There must be no conflicts in the set of all *ppRenderTargetViews* and *ppUnorderedAccessViews*.

- *ppDepthStencilView* must match the render-target views. For more information about resource views, see [Introduction to a Resource in Direct3D 11](#).

**OMSetRenderTargetsAndUnorderedAccessViews** performs the following tasks:

- Unbinds all currently bound conflicting resources (stream-output target resources (SOTargets), compute shader (CS) UAVs, shader-resource views (SRVs)).
- Binds *ppRenderTargetViews*, *ppDepthStencilView*, and *ppUnorderedAccessViews*.

## 2. *NumRTVs == D3D11\_KEEP\_RENDER\_TARGETS\_AND\_DEPTH\_STENCIL*

In this situation, **OMSetRenderTargetsAndUnorderedAccessViews** binds only UAVs.

The following conditions must be true for

**OMSetRenderTargetsAndUnorderedAccessViews** to succeed and for the runtime to pass the bind information to the driver:

- *UAVStartSlot + NumUAVs <= 8*
- There must be no conflicts in *ppUnorderedAccessViews*.

**OMSetRenderTargetsAndUnorderedAccessViews** unbinds the following items:

- All RTVs in slots  $>= \text{UAVStartSlot}$
- All RTVs that conflict with any UAVs in *ppUnorderedAccessViews*
- All currently bound resources (SOTargets, CS UAVs, SRVs) that conflict with *ppUnorderedAccessViews*

**OMSetRenderTargetsAndUnorderedAccessViews** binds *ppUnorderedAccessViews*.

**OMSetRenderTargetsAndUnorderedAccessViews** ignores *ppDepthStencilView*, and the current depth-stencil view remains bound.

## 3. *NumUAVs == D3D11\_KEEP\_UNORDERED\_ACCESS\_VIEWS*

In this situation, **OMSetRenderTargetsAndUnorderedAccessViews** binds only RTVs and DSV.

The following conditions must be true for

**OMSetRenderTargetsAndUnorderedAccessViews** to succeed and for the runtime to pass the bind information to the driver:

- *NumRTVs <= 8*

- There must be no conflicts in *ppRenderTargetViews*.
- *ppDepthStencilView* must match the render-target views. For more information about resource views, see [Introduction to a Resource in Direct3D 11](#).

**OMSetRenderTargetsAndUnorderedAccessViews** unbinds the following items:

- All UAVs in slots < *NumRTVs*
- All UAVs that conflict with any RTVs in *ppRenderTargetViews*
- All currently bound resources (SOTargets, CS UAVs, SRVs) that conflict with *ppRenderTargetViews*

**OMSetRenderTargetsAndUnorderedAccessViews** binds *ppRenderTargetViews* and *ppDepthStencilView*.

**OMSetRenderTargetsAndUnorderedAccessViews** ignores *UAVStartSlot*.

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::PSGetConstantBuffers method (d3d11.h)

Article 02/22/2024

Get the constant buffers used by the pixel shader pipeline stage.

## Syntax

C++

```
void PSGetConstantBuffers(  
    [in]          UINT      StartSlot,  
    [in]          UINT      NumBuffers,  
    [out, optional] ID3D11Buffer **ppConstantBuffers  
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - StartSlot).

[out, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*\\*](#)

Array of constant buffer interface pointers (see [ID3D11Buffer](#)) to be returned by the method.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::PSGetSamplers method (d3d11.h)

Article 02/22/2024

Get an array of sampler states from the pixel shader pipeline stage.

## Syntax

C++

```
void PSGetSamplers(
    [in]             UINT           StartSlot,
    [in]             UINT           NumSamplers,
    [out, optional] ID3D11SamplerState **ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into a zero-based array to begin getting samplers from (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers to get from a device context. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[out, optional] ppSamplers

Type: [ID3D11SamplerState\\*\\*](#)

Array of sampler-state interface pointers (see [ID3D11SamplerState](#)) to be returned by the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::PSGetShader method (d3d11.h)

Article 02/22/2024

Get the pixel shader currently set on the device.

## Syntax

C++

```
void PSGetShader(
    [out]           ID3D11PixelShader** ppPixelShader,
    [out, optional] ID3D11ClassInstance** ppClassInstances,
    [in, out, optional] UINT* pNumClassInstances
);
```

## Parameters

[out] ppPixelShader

Type: [ID3D11PixelShader\\*\\*](#)

Address of a pointer to a pixel shader (see [ID3D11PixelShader](#)) to be returned by the method.

[out, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*\\*](#)

Pointer to an array of class instance interfaces (see [ID3D11ClassInstance](#)).

[in, out, optional] pNumClassInstances

Type: [UINT\\*](#)

The number of class-instance elements in the array.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed, to avoid memory leaks.

**Windows Phone 8:** This API is supported.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?



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

# ID3D11DeviceContext::PSGetShaderResources method (d3d11.h)

Article 02/22/2024

Get the pixel shader resources.

## Syntax

C++

```
void PSGetShaderResources(
    [in]          UINT           StartSlot,
    [in]          UINT           NumViews,
    [out, optional] ID3D11ShaderResourceView **ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin getting shader resources from (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

The number of resources to get from the device. Up to a maximum of 128 slots are available for shader resources (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[out, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*\\*](#)

Array of [shader resource view](#) interfaces to be returned by the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::PSSetConstantBuffers method (d3d11.h)

Article 10/13/2021

Sets the constant buffers used by the pixel shader pipeline stage.

## Syntax

C++

```
void PSSetConstantBuffers(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting constant buffers to (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to set (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - *StartSlot*).

[in, optional] ppConstantBuffers

Type: [ID3D11Buffer](#)\*

Array of constant buffers (see [ID3D11Buffer](#)) being given to the device.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems, can bind a larger number of [ID3D11Buffer](#) resources to the shader than the maximum constant buffer size that is supported by shaders (4096 constants – 432-bit components each). *When you bind such a large buffer, the shader can access only the first 4096 432-bit component constants in the buffer, as if 4096 constants is the full size of the buffer.*

To enable the shader to access other parts of the buffer, call [PSSetConstantBuffers1](#) instead of [PSSetConstantBuffers](#). [PSSetConstantBuffers1](#) has additional parameters *pFirstConstant* and *pNumConstants*.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::PSSetSamplers method (d3d11.h)

Article 02/22/2024

Set an array of sampler states to the pixel shader pipeline stage.

## Syntax

C++

```
void PSSetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [in, optional] ID3D11SamplerState * const *ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting samplers to (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers in the array. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[in, optional] ppSamplers

Type: [ID3D11SamplerState\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)). See Remarks.

## Return value

None

## Remarks

Any sampler may be set to **NULL**; this invokes the default state, which is defined to be the following.

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

State	Default Value
Filter	D3D11_FILTER_MIN_MAG_MIP_LINEAR
AddressU	D3D11_TEXTURE_ADDRESS_CLAMP
AddressV	D3D11_TEXTURE_ADDRESS_CLAMP
AddressW	D3D11_TEXTURE_ADDRESS_CLAMP
MipLODBias	0
MaxAnisotropy	1
ComparisonFunc	D3D11_COMPARISON_NEVER
BorderColor[0]	1.0f
BorderColor[1]	1.0f
BorderColor[2]	1.0f
BorderColor[3]	1.0f
MinLOD	-FLT_MAX
MaxLOD	FLT_MAX

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::PSSetShader method (d3d11.h)

Article 02/22/2024

Sets a pixel shader to the device.

## Syntax

C++

```
void PSSetShader(  
    [in, optional] ID3D11PixelShader    *pPixelShader,  
    [in, optional] ID3D11ClassInstance * const *ppClassInstances,  
                           UINT           NumClassInstances  
) ;
```

## Parameters

[in, optional] pPixelShader

Type: [ID3D11PixelShader\\*](#)

Pointer to a pixel shader (see [ID3D11PixelShader](#)). Passing in **NULL** disables the shader for this pipeline stage.

[in, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*](#)

A pointer to an array of class-instance interfaces (see [ID3D11ClassInstance](#)). Each interface used by a shader must have a corresponding class instance or the shader will get disabled. Set ppClassInstances to **NULL** if the shader does not use any interfaces.

NumClassInstances

Type: [UINT](#)

The number of class-instance interfaces in the array.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The maximum number of instances a shader can have is 256.

Set `ppClassInstances` to **NULL** if no interfaces are used in the shader. If it is not **NULL**, the number of class instances must match the number of interfaces used in the shader. Furthermore, each interface pointer must have a corresponding class instance or the assigned shader will be disabled.

**Windows Phone 8:** This API is supported.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::PSSetShaderResources method (d3d11.h)

Article 02/22/2024

Bind an array of shader resources to the pixel shader stage.

## Syntax

C++

```
void PSSetShaderResources(
    [in]          UINT             StartSlot,
    [in]          UINT             NumViews,
    [in, optional] ID3D11ShaderResourceView * const *ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting shader resources to (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

Number of shader resources to set. Up to a maximum of 128 slots are available for shader resources (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[in, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*](#)

Array of [shader resource view](#) interfaces to set to the device.

## Return value

None

## Remarks

If an overlapping resource view is already bound to an output slot, such as a rendertarget, then this API will fill the destination shader resource slot with **NULL**.

For information about creating shader-resource views, see [ID3D11Device::CreateShaderResourceView](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::ResolveSubresource method (d3d11.h)

Article 10/13/2021

Copy a multisampled resource into a non-multisampled resource.

## Syntax

C++

```
void ResolveSubresource(
    [in] ID3D11Resource *pDstResource,
    [in] UINT           DstSubresource,
    [in] ID3D11Resource *pSrcResource,
    [in] UINT           SrcSubresource,
    [in] DXGI_FORMAT    Format
);
```

## Parameters

[in] pDstResource

Type: [ID3D11Resource\\*](#)

Destination resource. Must be created with the [D3D11\\_USAGE\\_DEFAULT](#) flag and be single-sampled. See [ID3D11Resource](#).

[in] DstSubresource

Type: [UINT](#)

A zero-based index, that identifies the destination subresource. Use [D3D11CalcSubresource](#) to calculate the index.

[in] pSrcResource

Type: [ID3D11Resource\\*](#)

Source resource. Must be multisampled.

[in] SrcSubresource

Type: [UINT](#)

The source subresource of the source resource.

[in] Format

Type: [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

This API is most useful when re-using the resulting rendertarget of one render pass as an input to a second render pass.

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"><li>• The source (or dest) format could be DXGI_FORMAT_R16G16B16A16_UNORM</li></ul>

- The dest (or source) format could be  
DXGI\_FORMAT\_R16G16B16A16\_FLOAT

# Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::RSGetScissorRects method (d3d11.h)

Article 02/22/2024

Get the array of scissor rectangles bound to the rasterizer stage.

## Syntax

C++

```
void RSGetScissorRects(  
    [in, out]        UINT      *pNumRects,  
    [out, optional] D3D11_RECT *pRects  
);
```

## Parameters

[in, out] pNumRects

Type: [UINT\\*](#)

The number of scissor rectangles (ranges between 0 and [D3D11\\_VIEWPORT\\_AND\\_SCISSORRECT\\_OBJECT\\_COUNT\\_PER\\_PIPELINE](#)) bound; set pRects to **NULL** to use pNumRects to see how many rectangles would be returned.

[out, optional] pRects

Type: [D3D11\\_RECT\\*](#)

An array of scissor rectangles (see [D3D11\\_RECT](#)). If NumRects is greater than the number of scissor rects currently bound, then unused members of the array will contain 0.

## Return value

None

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::RSGetState method (d3d11.h)

Article 02/22/2024

Get the [rasterizer state](#) from the rasterizer stage of the pipeline.

## Syntax

C++

```
void RSGetState(  
    [out] ID3D11RasterizerState **ppRasterizerState  
);
```

## Parameters

[out] ppRasterizerState

Type: [ID3D11RasterizerState\\*\\*](#)

Address of a pointer to a rasterizer-state interface (see [ID3D11RasterizerState](#)) to fill with information from the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call [IUnknown::Release](#) on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::RSGetViewports method (d3d11.h)

Article 02/22/2024

Gets the array of viewports bound to the rasterizer stage.

## Syntax

C++

```
void RSGetViewports(
    [in, out]        UINT          *pNumViewports,
    [out, optional]  D3D11_VIEWPORT *pViewports
);
```

## Parameters

[in, out] pNumViewports

Type: [UINT\\*](#)

A pointer to a variable that, on input, specifies the number of viewports (ranges from 0 to [D3D11\\_VIEWPORT\\_AND\\_SCISSORRECT\\_OBJECT\\_COUNT\\_PER\\_PIPELINE](#)) in the *pViewports* array; on output, the variable contains the actual number of viewports that are bound to the rasterizer stage. If *pViewports* is **NULL**, **RSGetViewports** fills the variable with the number of viewports currently bound.

**Note** In some versions of the Windows SDK, a [debug device](#) will raise an exception if the input value in the variable to which *pNumViewports* points is greater than [D3D11\\_VIEWPORT\\_AND\\_SCISSORRECT\\_OBJECT\\_COUNT\\_PER\\_PIPELINE](#) even if *pViewports* is **NULL**. The regular runtime ignores the value in the variable to which *pNumViewports* points when *pViewports* is **NULL**. This behavior of a debug device might be corrected in a future release of the Windows SDK.

[out, optional] pViewports

Type: [D3D11\\_VIEWPORT\\*](#)

An array of [D3D11\\_VIEWPORT](#) structures for the viewports that are bound to the rasterizer stage. If the number of viewports (in the variable to which *pNumViewports* points) is greater than the actual number of viewports currently bound, unused elements of the array contain 0. For info about how the viewport size depends on the device [feature level](#), which has changed between Direct3D 11 and Direct3D 10, see [D3D11\\_VIEWPORT](#).

## Return value

None

## Remarks

Windows Phone 8: This API is supported.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::RSSetScissorRects method (d3d11.h)

Article 02/22/2024

Bind an array of scissor rectangles to the rasterizer stage.

## Syntax

C++

```
void RSSetScissorRects(  
    [in]          UINT          NumRects,  
    [in, optional] const D3D11_RECT *pRects  
);
```

## Parameters

[in] NumRects

Type: **UINT**

Number of scissor rectangles to bind.

[in, optional] pRects

Type: **const D3D11\_RECT\***

An array of scissor rectangles (see [D3D11\\_RECT](#)).

## Return value

None

## Remarks

All scissor rects must be set atomically as one operation. Any scissor rects not defined by the call are disabled.

The scissor rectangles will only be used if ScissorEnable is set to true in the rasterizer state (see [D3D11\\_RASTERIZER\\_DESC](#)).

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 [ID3D11DeviceContext::RSSetViewports](#)).

**Windows Phone 8:** This API is supported.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::RSSetState method (d3d11.h)

Article 02/22/2024

Set the [rasterizer state](#) for the rasterizer stage of the pipeline.

## Syntax

C++

```
void RSSetState(  
    [in, optional] ID3D11RasterizerState *pRasterizerState  
);
```

## Parameters

[in, optional] pRasterizerState

Type: [ID3D11RasterizerState\\*](#)

Pointer to a rasterizer-state interface (see [ID3D11RasterizerState](#)) to bind to the pipeline.

## Return value

None

## Remarks

To create a rasterizer state interface, call [ID3D11Device::CreateRasterizerState](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::RSSetViewports method (d3d11.h)

Article 02/22/2024

Bind an array of viewports to the rasterizer stage of the pipeline.

## Syntax

C++

```
void RSSetViewports(
    [in]          UINT           NumViewports,
    [in, optional] const D3D11_VIEWPORT *pViewports
);
```

## Parameters

[in] NumViewports

Type: **UINT**

Number of viewports to bind.

[in, optional] pViewports

Type: **const D3D11\_VIEWPORT\***

An array of **D3D11\_VIEWPORT** structures to bind to the device. See the structure page for details about how the viewport size is dependent on the device feature level which has changed between Direct3D 11 and Direct3D 10.

## 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 [D3D11\\_VIEWPORT](#) structure for the *pViewports* array in a call to [ID3D11DeviceContext::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.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::SetPredication method (d3d11.h)

Article 07/27/2022

Set a rendering predicate.

## Syntax

C++

```
void SetPredication(
    [in, optional] ID3D11Predicate *pPredicate,
    [in]           BOOL          PredicateValue
);
```

## Parameters

[in, optional] pPredicate

Type: [ID3D11Predicate\\*](#)

A pointer to the [ID3D11Predicate](#) interface that represents the rendering predicate. A **NULL** value indicates "no" predication; in this case, the value of *PredicateValue* is irrelevant but will be preserved for [ID3D11DeviceContext::GetPredication](#).

[in] PredicateValue

Type: [BOOL](#)

If **TRUE**, rendering will be affected by when the predicate's conditions are met. If **FALSE**, rendering will be affected when the conditions are not met.

## Return value

None

## Remarks

The predicate must be in the "issued" or "signaled" state to be used for predication. While the predicate is set for predication, calls to [ID3D11DeviceContext::Begin](#) and

[ID3D11DeviceContext::End](#) are invalid.

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 *PredicateValue*. However, some predicates are only hints, so they may not actually prevent operations from being performed.

The primary usefulness of predication is to allow an application to issue rendering and resource manipulation commands without taking the performance hit of spinning, waiting for [ID3D11DeviceContext::GetData](#) to return. So, predication can occur while [ID3D11DeviceContext::GetData](#) returns **S\_FALSE**. Another way to think of it: an application can also use predication as a fallback, if it is possible that [ID3D11DeviceContext::GetData](#) returns **S\_FALSE**. If [ID3D11DeviceContext::GetData](#) returns **S\_OK**, the application can skip calling the rendering and resource manipulation commands manually with its own application logic.

Rendering and resource manipulation commands for Direct3D 11 include these Draw, Dispatch, Copy, Update, Clear, Generate, and Resolve operations.

- [Draw](#)
- [DrawAuto](#)
- [DrawIndexed](#)
- [DrawIndexedInstanced](#)
- [DrawIndexedInstancedIndirect](#)
- [DrawInstanced](#)
- [DrawInstancedIndirect](#)
- [Dispatch](#)
- [DispatchIndirect](#)
- [CopyResource](#)
- [CopyStructureCount](#)
- [CopySubresourceRegion](#)
- [CopySubresourceRegion1](#)
- [CopyTiles](#)
- [CopyTileMappings](#)
- [UpdateSubresource](#)
- [UpdateSubresource1](#)
- [UpdateTiles](#)
- [UpdateTileMappings](#)
- [ClearRenderTargetView](#)
- [ClearUnorderedAccessViewFloat](#)
- [ClearUnorderedAccessViewUint](#)
- [ClearView](#)

- [ClearDepthStencilView](#)
- [GenerateMips](#)
- [ResolveSubresource](#)

You can set a rendering predicate on an immediate or a deferred context. For info about immediate and deferred contexts, see [Immediate and Deferred Rendering](#).

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::SetResourceMinLOD method (d3d11.h)

Article 02/22/2024

Sets the minimum level-of-detail (LOD) for a resource.

## Syntax

C++

```
void SetResourceMinLOD(  
    [in] ID3D11Resource *pResource,  
    FLOAT             MinLOD  
)
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

A pointer to an [ID3D11Resource](#) that represents the resource.

MinLOD

Type: [FLOAT](#)

The level-of-detail, which ranges between 0 and the maximum number of mipmap levels of the resource. For example, the maximum number of mipmap levels of a 1D texture is specified in the [MipLevels](#) member of the [D3D11\\_TEXTURE1D\\_DESC](#) structure.

## Return value

None

## Remarks

To use a resource with [SetResourceMinLOD](#), you must set the [D3D11\\_RESOURCE\\_MISC\\_RESOURCE\\_CLAMP](#) flag when you create that resource.

For Direct3D 10 and Direct3D 10.1, when sampling from a texture resource in a shader, the sampler can define a minimum LOD clamp to force sampling from less detailed mip levels. For Direct3D 11, this functionality is extended from the sampler to the entire resource. Therefore, the application can specify the highest-resolution mip level of a resource that is available for access. This restricts the set of mip levels that are required to be resident in GPU memory, thereby saving memory.

The set of mip levels resident per-resource in GPU memory can be specified by the user.

Minimum LOD affects all of the resident mip levels. Therefore, only the resident mip levels can be updated and read from.

All methods that access texture resources must adhere to minimum LOD clamps.

Empty-set accesses are handled as out-of-bounds cases.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::SOGetTargets method (d3d11.h)

Article 10/13/2021

Get the target output buffers for the stream-output stage of the pipeline.

## Syntax

C++

```
void SOGetTargets(
    [in]          UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppSOTargets
);
```

## Parameters

[in] NumBuffers

Type: [UINT](#)

Number of buffers to get.

[out, optional] ppSOTargets

Type: [ID3D11Buffer\\*\\*](#)

An array of output buffers (see [ID3D11Buffer](#)) to be retrieved from the device.

## Return value

None

## Remarks

A maximum of four output buffers can be retrieved.

The offsets to the output buffers pointed to in the returned *ppSOTargets* array may be assumed to be -1 (append), as defined for use in [ID3D11DeviceContext::SOSetTargets](#).

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::SOSetTargets method (d3d11.h)

Article 11/23/2021

Set the target output buffers for the stream-output stage of the pipeline.

## Syntax

C++

```
void SOSetTargets(
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppSOTargets,
    [in, optional] const UINT   *pOffsets
);
```

## Parameters

[in] NumBuffers

Type: **UINT**

The number of buffer to bind to the device. A maximum of four output buffers can be set. If less than four are defined by the call, the remaining buffer slots are set to **NULL**. See Remarks.

[in, optional] ppSOTargets

Type: **ID3D11Buffer\***

The array of output buffers (see [ID3D11Buffer](#)) to bind to the device. The buffers must have been created with the [D3D11\\_BIND\\_STREAM\\_OUTPUT](#) flag.

[in, optional] pOffsets

Type: **const UINT\***

Array of offsets to the output buffers from *ppSOTargets*, one offset for each buffer. The offset values must be in bytes.

## Return value

None

## Remarks

An offset of -1 will cause the stream output buffer to be appended, continuing after the last location written to the buffer in a previous stream output pass.

Calling this method using a buffer that is currently bound for writing will effectively bind `NULL` instead because a buffer cannot be bound as both an input and an output at the same time.

The debug layer will generate a warning whenever a resource is prevented from being bound simultaneously as an input and an output, but this will not prevent invalid data from being used by the runtime.

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

Note that unlike some other resource methods in Direct3D, all currently bound targets will be unbound by calling `SOSetTargets(0, nullptr, nullptr);`.

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3D11DeviceContext::Unmap method (d3d11.h)

Article 02/22/2024

Invalidate the pointer to a resource and reenable the GPU's access to that resource.

## Syntax

C++

```
void Unmap(
    [in] ID3D11Resource *pResource,
    [in] UINT           Subresource
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

A pointer to a [ID3D11Resource](#) interface.

[in] Subresource

Type: [UINT](#)

A subresource to be unmapped.

## Return value

None

## Remarks

For info about how to use **Unmap**, see [How to: Use dynamic resources](#).

**Windows Phone 8:** This API is supported.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::UpdateSubresource method (d3d11.h)

Article 07/27/2022

See the [Basic hologram sample](#).

The CPU copies data from memory to a subresource created in non-mappable memory.

## Syntax

C++

```
void UpdateSubresource(
    [in]          ID3D11Resource  *pDstResource,
    [in]          UINT           DstSubresource,
    [in, optional] const D3D11_BOX *pDstBox,
    [in]          const void     *pSrcData,
    [in]          UINT           SrcRowPitch,
    [in]          UINT           SrcDepthPitch
);
```

## Parameters

[in] pDstResource

Type: [ID3D11Resource\\*](#)

A pointer to the destination resource (see [ID3D11Resource](#)).

[in] DstSubresource

Type: [UINT](#)

A zero-based index, that identifies the destination subresource. See [D3D11CalcSubresource](#) for more details.

[in, optional] pDstBox

Type: [const D3D11\\_BOX\\*](#)

A pointer to a box that defines the portion of the destination subresource to copy the resource data into. Coordinates are in bytes for buffers and in texels for textures. If

**NULL**, the data is written to the destination subresource with no offset. The dimensions of the source must fit the destination (see [D3D11\\_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, **UpdateSubresource** doesn't perform an update operation.

[in] *pSrcData*

Type: **const void\***

A pointer to the source data in memory.

[in] *SrcRowPitch*

Type: **UINT**

The size of one row of the source data.

[in] *SrcDepthPitch*

Type: **UINT**

The size of one depth slice of source data.

## Return value

None

## Remarks

For a shader-constant buffer; set *pDstBox* to **NULL**. It is not possible to use this method to partially update a shader-constant buffer.

A resource cannot be used as a destination if:

- the resource is created with [immutable](#) or [dynamic](#) usage.
- the resource is created as a depth-stencil resource.
- the resource is created with multisampling capability (see [DXGI\\_SAMPLE\\_DESC](#)).

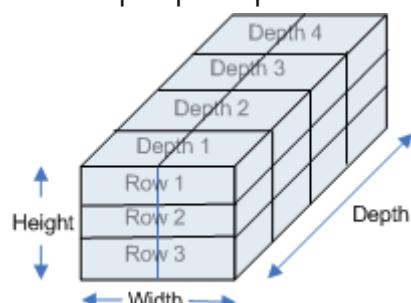
When **UpdateSubresource** returns, the application is free to change or even free the data pointed to by *pSrcData* because the method has already copied/snapped away the original contents.

The performance of **UpdateSubresource** depends on whether or not there is contention for the destination resource. For example, contention for a vertex buffer resource occurs when the application executes a **Draw** call and later calls **UpdateSubresource** on the same vertex buffer before the **Draw** call is actually executed by the GPU.

- When there is contention for the resource, **UpdateSubresource** will perform 2 copies of the source data. First, the data is copied by the CPU to a temporary storage space accessible by the command buffer. This copy happens before the method returns. A second copy is then performed by the GPU to copy the source data into non-mappable memory. This second copy happens asynchronously because it is executed by GPU when the command buffer is flushed.
- When there is no resource contention, the behavior of **UpdateSubresource** is dependent on which is faster (from the CPU's perspective): copying the data to the command buffer and then having a second copy execute when the command buffer is flushed, or having the CPU copy the data to the final resource location. This is dependent on the architecture of the underlying system.

**Note** Applies only to feature level 9\_x hardware If you use **UpdateSubresource** or **ID3D11DeviceContext::CopySubresourceRegion** to copy from a staging resource to a default resource, you can corrupt the destination contents. This occurs if you pass a **NULL** source box and if the source resource has different dimensions from those of the destination resource or if you use destination offsets, (x, y, and z). In this situation, always pass a source box that is the full size of the source resource.

To better understand the source row pitch and source depth pitch parameters, the



following illustration shows a 3D volume texture.

Each block in this visual represents an element of data, and the size of each element is dependent on the resource's format. For example, if the resource format is **DXGI\_FORMAT\_R32G32B32A32\_FLOAT**, the size of each element would be 128 bits, or 16 bytes. This 3D volume texture has a width of two, a height of three, and a depth of four.

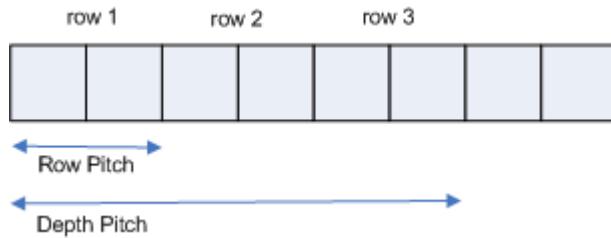
To calculate the source row pitch and source depth pitch for a given resource, use the following formulas:

- Source Row Pitch = [size of one element in bytes] \* [number of elements in one row]
- Source Depth Pitch = [Source Row Pitch] \* [number of rows (height)]

In the case of this example 3D volume texture where the size of each element is 16 bytes, the formulas are as follows:

- Source Row Pitch =  $16 * 2 = 32$
- Source Depth Pitch =  $16 * 2 * 3 = 96$

The following illustration shows the resource as it is laid out in memory.



For example, the following code snippet shows how to specify a destination region in a 2D texture. Assume the destination texture is 512x512 and the operation will copy the data pointed to by *pData* to [(120,100)..(200,220)] in the destination texture. Also assume that *rowPitch* has been initialized with the proper value (as explained above). **front** and **back** are set to 0 and 1 respectively, because by having **front** equal to **back**, the box is technically empty.

```
D3D11_BOX destRegion;
destRegion.left = 120;
destRegion.right = 200;
destRegion.top = 100;
destRegion.bottom = 220;
destRegion.front = 0;
destRegion.back = 1;

pd3dDeviceContext->UpdateSubresource( pDestTexture, 0, &destRegion, pData,
rowPitch, 0 );
```

The 1D case is similar. The following snippet shows how to specify a destination region in a 1D texture. Use the same assumptions as above, except that the texture is 512 in length.

```

D3D11_BOX destRegion;
destRegion.left = 120;
destRegion.right = 200;
destRegion.top = 0;
destRegion.bottom = 1;
destRegion.front = 0;
destRegion.back = 1;

pd3dDeviceContext->UpdateSubresource( pDestTexture, 0, &destRegion, pData,
rowPitch, 0 );

```

For info about various resource types and how `UpdateSubresource` might work with each resource type, see [Introduction to a Resource in Direct3D 11](#).

## Calling `UpdateSubresource` on a Deferred Context

If your application calls `UpdateSubresource` on a deferred context with a destination box—to which *pDstBox* points—that has a non-(0,0,0) offset, and if the driver does not support command lists, `UpdateSubresource` inappropriately applies that destination-box offset to the *pSrcData* parameter. To work around this behavior, use the following code:

```

HRESULT UpdateSubresource_Workaround(
    ID3D11Device *pDevice,
    ID3D11DeviceContext *pDeviceContext,
    ID3D11Resource *pDstResource,
    UINT dstSubresource,
    const D3D11_BOX *pDstBox,
    const void *pSrcData,
    UINT srcBytesPerElement,
    UINT srcRowPitch,
    UINT srcDepthPitch,
    bool* pDidWorkAround )
{
    HRESULT hr = S_OK;
    bool needWorkaround = false;
    D3D11_DEVICE_CONTEXT_TYPE contextType = pDeviceContext->GetType();

    if( pDstBox && (D3D11_DEVICE_CONTEXT_DEFERRED == contextType) )
    {
        D3D11_FEATURE_DATA_THREADING threadingCaps = { FALSE, FALSE };

        hr = pDevice->CheckFeatureSupport( D3D11_FEATURE_THREADING,
&threadingCaps, sizeof(threadingCaps) );
    }
}

```

```

        if( SUCCEEDED(hr) )
        {
            if( !threadingCaps.DriverCommandLists )
            {
                needWorkaround = true;
            }
        }

const void* pAdjustedSrcData = pSrcData;

if( needWorkaround )
{
    D3D11_BOX alignedBox = *pDstBox;

    // convert from pixels to blocks
    if( m_bBC )
    {
        alignedBox.left      /= 4;
        alignedBox.right     /= 4;
        alignedBox.top       /= 4;
        alignedBox.bottom    /= 4;
    }

    pAdjustedSrcData = ((const BYTE*)pSrcData) - (alignedBox.front * srcDepthPitch) - (alignedBox.top * srcRowPitch) - (alignedBox.left * srcBytesPerElement);
}

pDeviceContext->UpdateSubresource( pDstResource, dstSubresource,
pDstBox, pAdjustedSrcData, srcRowPitch, srcDepthPitch );

if( pDidWorkAround )
{
    *pDidWorkAround = needWorkaround;
}

return hr;
}

```

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

[ID3D11Resource](#)

[Basic hologram sample ↗](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::VSGetConstantBuffers method (d3d11.h)

Article 07/27/2022

Get the constant buffers used by the vertex shader pipeline stage.

## Syntax

C++

```
void VSGetConstantBuffers(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - StartSlot).

[out, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*\\*](#)

Array of constant buffer interface pointers (see [ID3D11Buffer](#)) to be returned by the method.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::VSGetSamplers method (d3d11.h)

Article 02/22/2024

Get an array of sampler states from the vertex shader pipeline stage.

## Syntax

C++

```
void VSGetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [out, optional] ID3D11SamplerState **ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into a zero-based array to begin getting samplers from (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers to get from a device context. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[out, optional] ppSamplers

Type: [ID3D11SamplerState\\*\\*](#)

Array of sampler-state interface pointers (see [ID3D11SamplerState](#)) to be returned by the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::VSGetShader method (d3d11.h)

Article 02/22/2024

Get the vertex shader currently set on the device.

## Syntax

C++

```
void VSGetShader(
    [out]           ID3D11VertexShader  **ppVertexShader,
    [out, optional] ID3D11ClassInstance **ppClassInstances,
    [in, out, optional] UINT          *pNumClassInstances
);
```

## Parameters

[out] ppVertexShader

Type: [ID3D11VertexShader\\*\\*](#)

Address of a pointer to a vertex shader (see [ID3D11VertexShader](#)) to be returned by the method.

[out, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*\\*](#)

Pointer to an array of class instance interfaces (see [ID3D11ClassInstance](#)).

[in, out, optional] pNumClassInstances

Type: [UINT\\*](#)

The number of class-instance elements in the array.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::VSGetShaderResources method (d3d11.h)

Article 02/22/2024

Get the vertex shader resources.

## Syntax

C++

```
void VSGetShaderResources(
    [in]          UINT           StartSlot,
    [in]          UINT           NumViews,
    [out, optional] ID3D11ShaderResourceView **ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin getting shader resources from (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

The number of resources to get from the device. Up to a maximum of 128 slots are available for shader resources (ranges from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[out, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*\\*](#)

Array of [shader resource view](#) interfaces to be returned by the device.

## Return value

None

## Remarks

Any returned interfaces will have their reference count incremented by one. Applications should call `IUnknown::Release` on the returned interfaces when they are no longer needed to avoid memory leaks.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::VSSetConstantBuffers method (d3d11.h)

Article 10/13/2021

Sets the constant buffers used by the vertex shader pipeline stage.

## Syntax

C++

```
void VSSetConstantBuffers(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting constant buffers to (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to set (ranges from 0 to [D3D11\\_COMMONSHADER\\_CONSTANT\\_BUFFER\\_API\\_SLOT\\_COUNT](#) - *StartSlot*).

[in, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*](#)

Array of constant buffers (see [ID3D11Buffer](#)) being given to the device.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The Direct3D 11.1 runtime, which is available starting with Windows 8, can bind a larger number of [ID3D11Buffer](#) resources to the shader than the maximum constant buffer size that is supported by shaders (4096 constants – 4\*32-bit components each). When you bind such a large buffer, the shader can access only the first 4096 4\*32-bit component constants in the buffer, as if 4096 constants is the full size of the buffer.

If the application wants the shader to access other parts of the buffer, it must call the [VSSetConstantBuffers1](#) method instead.

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::VSSetSamplers method (d3d11.h)

Article 02/22/2024

Set an array of sampler states to the vertex shader pipeline stage.

## Syntax

C++

```
void VSSetSamplers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumSamplers,
    [in, optional] ID3D11SamplerState * const *ppSamplers
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting samplers to (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - 1).

[in] NumSamplers

Type: [UINT](#)

Number of samplers in the array. Each pipeline stage has a total of 16 sampler slots available (ranges from 0 to D3D11\_COMMONSHADER\_SAMPLER\_SLOT\_COUNT - StartSlot).

[in, optional] ppSamplers

Type: [ID3D11SamplerState\\*](#)

Pointer to an array of sampler-state interfaces (see [ID3D11SamplerState](#)). See Remarks.

## Return value

None

## Remarks

Any sampler may be set to **NULL**; this invokes the default state, which is defined to be the following.

```
//Default sampler state:  
D3D11_SAMPLER_DESC SamplerDesc;  
SamplerDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;  
SamplerDesc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;  
SamplerDesc.MipLODBias = 0;  
SamplerDesc.MaxAnisotropy = 1;  
SamplerDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;  
SamplerDesc.BorderColor[0] = 1.0f;  
SamplerDesc.BorderColor[1] = 1.0f;  
SamplerDesc.BorderColor[2] = 1.0f;  
SamplerDesc.BorderColor[3] = 1.0f;  
SamplerDesc.MinLOD = -FLT_MAX;  
SamplerDesc.MaxLOD = FLT_MAX;
```

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::VSSetShader method (d3d11.h)

Article 02/22/2024

Set a vertex shader to the device.

## Syntax

C++

```
void VSSetShader(  
    [in, optional] ID3D11VertexShader  *pVertexShader,  
    [in, optional] ID3D11ClassInstance * const *ppClassInstances,  
                           UINT           NumClassInstances  
) ;
```

## Parameters

[in, optional] pVertexShader

Type: [ID3D11VertexShader\\*](#)

Pointer to a vertex shader (see [ID3D11VertexShader](#)). Passing in **NULL** disables the shader for this pipeline stage.

[in, optional] ppClassInstances

Type: [ID3D11ClassInstance\\*](#)

A pointer to an array of class-instance interfaces (see [ID3D11ClassInstance](#)). Each interface used by a shader must have a corresponding class instance or the shader will get disabled. Set ppClassInstances to **NULL** if the shader does not use any interfaces.

NumClassInstances

Type: [UINT](#)

The number of class-instance interfaces in the array.

## Return value

None

## Remarks

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

The maximum number of instances a shader can have is 256.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext::VSSetShaderResources method (d3d11.h)

Article 02/22/2024

Bind an array of shader resources to the vertex-shader stage.

## Syntax

C++

```
void VSSetShaderResources(
    [in]          UINT             StartSlot,
    [in]          UINT             NumViews,
    [in, optional] ID3D11ShaderResourceView * const *ppShaderResourceViews
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting shader resources to (range is from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - 1).

[in] NumViews

Type: [UINT](#)

Number of shader resources to set. Up to a maximum of 128 slots are available for shader resources (range is from 0 to D3D11\_COMMONSHADER\_INPUT\_RESOURCE\_SLOT\_COUNT - StartSlot).

[in, optional] ppShaderResourceViews

Type: [ID3D11ShaderResourceView\\*](#)

Array of [shader resource view](#) interfaces to set to the device.

## Return value

None

## Remarks

If an overlapping resource view is already bound to an output slot, such as a rendertarget, then this API will fill the destination shader resource slot with **NULL**.

For information about creating shader-resource views, see [ID3D11Device::CreateShaderResourceView](#).

The method will hold a reference to the interfaces passed in. This differs from the device state behavior in Direct3D 10.

In order to unbind resource slots, you must pass an array containing null values. For example, to clear the first 4 slots, use:

```
ID3D11ShaderResourceView* nullsrv[] = { nullptr, nullptr, nullptr, nullptr };
context->VSSetShaderResources(0, 4, nullsrv);
```

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1 interface (d3d11\_1.h)

Article 07/27/2022

The device context interface represents a device context; it is used to render commands. **ID3D11DeviceContext1** adds new methods to those in [ID3D11DeviceContext](#).

## Inheritance

The **ID3D11DeviceContext1** interface inherits from [ID3D11DeviceContext](#).

**ID3D11DeviceContext1** also has these types of members:

## Methods

The **ID3D11DeviceContext1** interface has these methods.

<a href="#">ID3D11DeviceContext1::ClearView</a>
Sets all the elements in a resource view to one value.
<a href="#">ID3D11DeviceContext1::CopySubresourceRegion1</a>
Copies a region from a source resource to a destination resource.
<a href="#">ID3D11DeviceContext1::CSGetConstantBuffers1</a>
Gets the constant buffers that the compute-shader stage uses.
<a href="#">ID3D11DeviceContext1::CSSetConstantBuffers1</a>
Sets the constant buffers that the compute-shader stage uses.
<a href="#">ID3D11DeviceContext1::DiscardResource</a>
Discards a resource from the device context.
<a href="#">ID3D11DeviceContext1::DiscardView</a>
Discards a resource view from the device context.

## [ID3D11DeviceContext1::DiscardView1](#)

Discards the specified elements in a resource view from the device context.

## [ID3D11DeviceContext1::DSGetConstantBuffers1](#)

Gets the constant buffers that the domain-shader stage uses.

## [ID3D11DeviceContext1::DSSetConstantBuffers1](#)

Sets the constant buffers that the domain-shader stage uses.

## [ID3D11DeviceContext1::GSGetConstantBuffers1](#)

Gets the constant buffers that the geometry shader pipeline stage uses.

## [ID3D11DeviceContext1::GSSetConstantBuffers1](#)

Sets the constant buffers that the geometry shader pipeline stage uses.

## [ID3D11DeviceContext1::HSGetConstantBuffers1](#)

Gets the constant buffers that the hull-shader stage uses.

## [ID3D11DeviceContext1::HSSetConstantBuffers1](#)

Sets the constant buffers that the hull-shader stage of the pipeline uses.

## [ID3D11DeviceContext1::PSGetConstantBuffers1](#)

Gets the constant buffers that the pixel shader pipeline stage uses.

## [ID3D11DeviceContext1::PSSetConstantBuffers1](#)

Sets the constant buffers that the pixel shader pipeline stage uses, and enables the shader to access other parts of the buffer.

## [ID3D11DeviceContext1::SwapDeviceContextState](#)

Activates the given context state object and changes the current device behavior to Direct3D 11, Direct3D 10.1, or Direct3D 10.

## [ID3D11DeviceContext1::UpdateSubresource1](#)

The CPU copies data from memory to a subresource created in non-mappable memory.  
(`ID3D11DeviceContext1.UpdateSubresource1`)

## [ID3D11DeviceContext1::VSGetConstantBuffers1](#)

Gets the constant buffers that the vertex shader pipeline stage uses.

## [ID3D11DeviceContext1::VSSetConstantBuffers1](#)

Sets the constant buffers that the vertex shader pipeline stage uses.

# Requirements

<b>Minimum supported client</b>	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
<b>Minimum supported server</b>	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
<b>Target Platform</b>	Windows
<b>Header</b>	d3d11_1.h

## See also

[Core Interfaces](#)

[ID3D11DeviceContext](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::ClearView method (d3d11\_1.h)

Article 10/13/2021

Sets all the elements in a resource view to one value.

## Syntax

C++

```
void ClearView(  
    [in]          ID3D11View      *pView,  
    [in]          const FLOAT [4]  Color,  
    [in, optional] const D3D11_RECT *pRect,  
                           UINT        NumRects  
) ;
```

## Parameters

[in] pView

A pointer to the [ID3D11View](#) interface that represents the resource view to clear.

[in] Color

A 4-component array that represents the color to use to clear the resource view.

[in, optional] pRect

An array of [D3D11\\_RECT](#) structures for the rectangles in the resource view to clear. If **NULL**, **ClearView** clears the entire surface.

NumRects

Number of rectangles in the array that the *pRect* parameter specifies.

## Return value

None

## Remarks

**ClearView** works only on render-target views (RTVs), depth/stencil views (DSVs) on depth-only resources (resources with no stencil component), unordered-access views (UAVs), or any video view of a [Texture2D](#) surface. The runtime drops invalid calls. Empty rectangles in the *pRect* array are a no-op. A rectangle is empty if the top value equals the bottom value or the left value equals the right value.

**ClearView** doesn't support 3D textures.

**ClearView** applies the same color value to all array slices in a view; all rectangles in the *pRect* array correspond to each array slice. The *pRect* array of rectangles is a set of areas to clear on a single surface. If the view is an array, **ClearView** clears all the rectangles on each array slice individually.

When you apply rectangles to buffers, set the top value to 0 and the bottom value to 1 and set the left value and right value to describe the extent within the buffer. When the top value equals the bottom value or the left value equals the right value, the rectangle is empty and a no-op is achieved.

The driver converts and clamps color values to the destination format as appropriate per Direct3D conversion rules. For example, if the format of the view is [DXGI\\_FORMAT\\_R8G8B8A8\\_UNORM](#), the driver clamps inputs to 0.0f to 1.0f (+INF -> 1.0f (0xFF)/NaN -> 0.0f).

If the format is integer, such as [DXGI\\_FORMAT\\_R8G8B8A8\\_UINT](#), the runtime interprets inputs as integral floats. Therefore, 235.0f maps to 235 (rounds to zero, out of range/INF values clamp to target range, and NaN to 0).

Here are the color mappings:

- Color[0]: R (or Y for video)
- Color[1]: G (or U/Cb for video)
- Color[2]: B (or V/Cr for video)
- Color[3]: A

For video views with YUV or YCbBr formats, **ClearView** doesn't convert color values. In situations where the format name doesn't indicate \_UNORM, \_UINT, and so on, **ClearView** assumes \_UINT. Therefore, 235.0f maps to 235 (rounds to zero, out of range/INF values clamp to target range, and NaN to 0).

## Requirements

---

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::CopySubresourceRegion1 method (d3d11\_1.h)

Article 10/13/2021

Copies a region from a source resource to a destination resource.

## Syntax

C++

```
void CopySubresourceRegion1(
    [in]          ID3D11Resource  *pDstResource,
    [in]          UINT           DstSubresource,
    [in]          UINT           DstX,
    [in]          UINT           DstY,
    [in]          UINT           DstZ,
    [in]          ID3D11Resource  *pSrcResource,
    [in]          UINT           SrcSubresource,
    [in, optional] const D3D11_BOX *pSrcBox,
    [in]          UINT           CopyFlags
);
```

## Parameters

[in] pDstResource

Type: [ID3D11Resource\\*](#)

A pointer to the destination resource.

[in] DstSubresource

Type: [UINT](#)

Destination subresource index.

[in] DstX

Type: [UINT](#)

The x-coordinate of the upper-left corner of the destination region.

[in] DstY

Type: **UINT**

The y-coordinate of the upper-left corner of the destination region. For a 1D subresource, this must be zero.

[in] DstZ

Type: **UINT**

The z-coordinate of the upper-left corner of the destination region. For a 1D or 2D subresource, this must be zero.

[in] pSrcResource

Type: **ID3D11Resource\***

A pointer to the source resource.

[in] SrcSubresource

Type: **UINT**

Source subresource index.

[in, optional] pSrcBox

Type: **const D3D11\_BOX\***

A pointer to a 3D box that defines the region of the source subresource that **CopySubresourceRegion1** can copy. If **NULL**, **CopySubresourceRegion1** copies the entire source subresource. The box must fit within the source resource.

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, **CopySubresourceRegion1** doesn't perform a copy operation.

[in] CopyFlags

Type: **UINT**

A **D3D11\_COPY\_FLAGS**-typed value that specifies how to perform the copy operation. If you specify zero for no copy option, **CopySubresourceRegion1** behaves like **ID3D11DeviceContext::CopySubresourceRegion**. For existing display drivers that can't process these flags, the runtime doesn't use them.

# Return value

None

## Remarks

If the display driver supports overlapping, the source and destination subresources can be identical, and the source and destination regions can overlap each other. For existing display drivers that don't support overlapping, the runtime drops calls with identical source and destination subresources, regardless of whether the regions overlap. To determine whether the display driver supports overlapping, check the **CopyWithOverlap** member of [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#). This overlapping support enables additional scroll functionality in a call to [IDXGISwapChain::Present](#).

**Note** Applies only to feature level 9\_x hardware If you use [ID3D11DeviceContext1::UpdateSubresource1](#) or [CopySubresourceRegion1](#) to copy from a staging resource to a default resource, you can corrupt the destination contents. This occurs if you pass a **NULL** source box and if the source resource has different dimensions from those of the destination resource or if you use destination offsets, (x, y, and z). In this situation, always pass a source box that is the full size of the source resource.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::CSGetConstantBuffers1 method (d3d11\_1.h)

Article 02/22/2024

Gets the constant buffers that the compute-shader stage uses.

## Syntax

C++

```
void CSGetConstantBuffers1(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers,
    [out, optional] UINT      *pFirstConstant,
    [out, optional] UINT      *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[out, optional] ppConstantBuffers

Array of constant buffer interface pointers to be returned by the method.

[out, optional] pFirstConstant

A pointer to an array that receives the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 2 indicates that the start of the associated constant

buffer is 32 bytes into the constant buffer. The runtime sets *pFirstConstant* to **NULL** if the buffers do not have offsets.

[out, optional] *pNumConstants*

A pointer to an array that receives the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. The runtime sets *pNumConstants* to **NULL** if it doesn't specify the numbers of constants in each buffer.

## Return value

None

## Remarks

If no buffer is bound at a slot, *pFirstConstant* and *pNumConstants* are **NULL** for that slot.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::CSSetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Sets the constant buffers that the compute-shader stage uses.

## Syntax

C++

```
void CSSetConstantBuffers1(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers,
    [in, optional] const UINT   *pFirstConstant,
    [in, optional] const UINT   *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the zero-based array to begin setting constant buffers to (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to set (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[in, optional] ppConstantBuffers

Array of constant buffers (see [ID3D11Buffer](#)) being given to the device.

[in, optional] pFirstConstant

An array that holds the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 16 indicates that the start of the associated constant buffer is 256 bytes into the constant buffer. Each offset must be a multiple of 16 constants.

[in, optional] *pNumConstants*

An array that holds the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. Each number of constants must be a multiple of 16 constants, in the range [0..4096].

## Return value

None

## Remarks

The runtime drops the call to **CSSetConstantBuffers1** if the number of constants to which *pNumConstants* points is larger than the maximum constant buffer size that is supported by shaders (4096 constants). The values in the elements of the *pFirstConstant* and *pFirstConstant + pNumConstants* arrays can exceed the length of each buffer; from the shader's point of view, the constant buffer is the intersection of the actual memory allocation for the buffer and the window [value in an element of *pFirstConstant*, value in an element of *pFirstConstant + pNumConstants*]. The runtime also drops the call to **CSSetConstantBuffers1** on existing drivers that don't support this offsetting.

The runtime will emulate this feature for [feature level](#) 9.1, 9.2, and 9.3; therefore, this feature is supported for feature level 9.1, 9.2, and 9.3. This feature is always available on new drivers for feature level 10 and higher.

From the shader's point of view, element [0] in the constant buffers array is the constant at *pFirstConstant*.

Out of bounds access to the constant buffers from the shader to the range that is defined by *pFirstConstant* and *pNumConstants* returns 0.

If *pFirstConstant* and *pNumConstants* arrays are **NULL**, you get the same result as if you were binding the entire buffer into view. You get this same result if you call the [CSSetConstantBuffers](#) method. If the buffer is larger than the maximum constant buffer size that is supported by shaders (4096 elements), the shader can access only the first 4096 constants.

If either *pFirstConstant* or *pNumConstants* is **NULL**, the other parameter must also be **NULL**.

# Calling `CSSetConstantBuffers1` with command list emulation

The runtime's [command list](#) emulation of `CSSetConstantBuffers1` sometimes doesn't actually change the offsets or sizes for the arrays of constant buffers. This behavior occurs when

`CSSetConstantBuffers1` doesn't effectively change the constant buffers at the beginning and end of the range of slots that you set to update. This section shows how to work around this

behavior.

Here is the code to check whether either the runtime emulates command lists or the driver supports command lists:

C++

```
HRESULT hr = S_OK;
bool needWorkaround = false;
D3D11_DEVICE_CONTEXT_TYPE contextType = pDeviceContext->GetType();

if( D3D11_DEVICE_CONTEXT_DEFERRED == contextType )
{
    D3D11_FEATURE_DATA_THREADING threadingCaps = { FALSE, FALSE };

    hr = pDevice->CheckFeatureSupport( D3D11_FEATURE_THREADING,
&threadingCaps, sizeof(threadingCaps) );
    if( SUCCEEDED(hr) && !threadingCaps.DriverCommandLists )
    {
        needWorkaround = true; // the runtime emulates command lists.
    }
}
```

If the runtime emulates command lists, you need to use one of these code snippets:

If you change the offset and size on only a single constant buffer, set the constant buffer to **NULL** first:

C++

```
pDeviceContext->CSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
if( needWorkaround )
{
    // Workaround for command list emulation
```

```
    pDeviceContext->CSSetConstantBuffers(0, 1, &NullCBuf);  
}  
pDeviceContext->CSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
```

If you change multiple constant buffers, set the first and last constant buffers of the range to **NULL** first:

C++

```
pDeviceContext->CSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);  
if( needWorkaround )  
{  
    // Workaround for command list emulation  
    pDeviceContext->CSSetConstantBuffers(0, 1, &NullCBuf);  
    pDeviceContext->CSSetConstantBuffers(3, 1, &NullCBuf);  
}  
pDeviceContext->CSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);
```

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3D11DeviceContext1::DiscardResource method (d3d11\_1.h)

Article 02/22/2024

Discards a resource from the device context.

## Syntax

C++

```
void DiscardResource(  
    [in] ID3D11Resource *pResource  
);
```

## Parameters

[in] pResource

Type: [ID3D11Resource\\*](#)

A pointer to the [ID3D11Resource](#) interface for the resource to discard. The resource must have been created with usage [D3D11\\_USAGE\\_DEFAULT](#) or [D3D11\\_USAGE\\_DYNAMIC](#), otherwise the runtime drops the call to `DiscardResource`; if the debug layer is enabled, the runtime returns an error message.

## Return value

None

## Remarks

`DiscardResource` informs the graphics processing unit (GPU) that the existing content in the resource that *pResource* points to is no longer needed.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::DiscardView method (d3d11\_1.h)

Article 02/22/2024

Discards a resource view from the device context.

## Syntax

C++

```
void DiscardView(  
    [in] ID3D11View *pResourceView  
);
```

## Parameters

[in] pResourceView

Type: [ID3D11View\\*](#)

A pointer to the [ID3D11View](#) interface for the resource view to discard. The resource that underlies the view must have been created with usage [D3D11\\_USAGE\\_DEFAULT](#) or [D3D11\\_USAGE\\_DYNAMIC](#), otherwise the runtime drops the call to **DiscardView**; if the debug layer is enabled, the runtime returns an error message.

## Return value

None

## Remarks

**DiscardView** informs the graphics processing unit (GPU) that the existing content in the resource view that *pResourceView* points to is no longer needed. The view can be an SRV, RTV, UAV, or DSV. **DiscardView** is a variation on the [DiscardResource](#) method. **DiscardView** allows you to discard a subset of a resource that is in a view (such as a single miplevel). More importantly, **DiscardView** provides a convenience because often views are what are being bound and unbound at the pipeline. Some pipeline bindings

do not have views, such as stream output. In that situation, `DiscardResource` can do the job for any resource.

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::DiscardView1 method (d3d11\_1.h)

Article 10/13/2021

Discards the specified elements in a resource view from the device context.

## Syntax

C++

```
void DiscardView1(
    [in]          ID3D11View      *pResourceView,
    [in, optional] const D3D11_RECT *pRects,
                           UINT        NumRects
);
```

## Parameters

[in] pResourceView

Type: [ID3D11View\\*](#)

A pointer to the [ID3D11View](#) interface for the resource view to discard. The resource that underlies the view must have been created with usage [D3D11\\_USAGE\\_DEFAULT](#) or [D3D11\\_USAGE\\_DYNAMIC](#), otherwise the runtime drops the call to [DiscardView1](#); if the debug layer is enabled, the runtime returns an error message.

[in, optional] pRects

Type: [const D3D11\\_RECT\\*](#)

An array of [D3D11\\_RECT](#) structures for the rectangles in the resource view to discard. If [NULL](#), [DiscardView1](#) discards the entire view and behaves the same as [DiscardView](#).

NumRects

Type: [UINT](#)

Number of rectangles in the array that the *pRects* parameter specifies.

## Return value

None

## Remarks

**DiscardView1** informs the graphics processing unit (GPU) that the existing content in the specified elements in the resource view that *pResourceView* points to is no longer needed. The view can be an SRV, RTV, UAV, or DSV. **DiscardView1** is a variation on the [DiscardResource](#) method. **DiscardView1** allows you to discard elements of a subset of a resource that is in a view (such as elements of a single miplevel). More importantly, **DiscardView1** provides a convenience because often views are what are being bound and unbound at the pipeline. Some pipeline bindings do not have views, such as stream output. In that situation, **DiscardResource** can do the job for any resource.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::DSGetConstantBuffers1 method (d3d11\_1.h)

Article 02/22/2024

Gets the constant buffers that the domain-shader stage uses.

## Syntax

C++

```
void DSGetConstantBuffers1(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers,
    [out, optional] UINT      *pFirstConstant,
    [out, optional] UINT      *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[out, optional] ppConstantBuffers

Array of constant buffer interface pointers to be returned by the method.

[out, optional] pFirstConstant

A pointer to an array that receives the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 2 indicates that the start of the associated constant

buffer is 32 bytes into the constant buffer. The runtime sets *pFirstConstant* to **NULL** if the buffers do not have offsets.

[out, optional] *pNumConstants*

A pointer to an array that receives the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. The runtime sets *pNumConstants* to **NULL** if it doesn't specify the numbers of constants in each buffer.

## Return value

None

## Remarks

If no buffer is bound at a slot, *pFirstConstant* and *pNumConstants* are **NULL** for that slot.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::DSSetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Sets the constant buffers that the domain-shader stage uses.

## Syntax

C++

```
void DSSetConstantBuffers1(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers,
    [in, optional] const UINT   *pFirstConstant,
    [in, optional] const UINT   *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the zero-based array to begin setting constant buffers to (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to set (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[in, optional] ppConstantBuffers

Array of constant buffers being given to the device.

[in, optional] pFirstConstant

An array that holds the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 16 indicates that the start of the associated constant buffer is 256 bytes into the constant buffer. Each offset must be a multiple of 16 constants.

[in, optional] *pNumConstants*

An array that holds the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. Each number of constants must be a multiple of 16 constants, in the range [0..4096].

## Return value

None

## Remarks

The runtime drops the call to **DSSetConstantBuffers1** if the number of constants to which *pNumConstants* points is larger than the maximum constant buffer size that is supported by shaders (4096 constants). The values in the elements of the *pFirstConstant* and *pFirstConstant + pNumConstants* arrays can exceed the length of each buffer; from the shader's point of view, the constant buffer is the intersection of the actual memory allocation for the buffer and the window [value in an element of *pFirstConstant*, value in an element of *pFirstConstant + pNumConstants*]. The runtime also drops the call to **DSSetConstantBuffers1** on existing drivers that don't support this offsetting.

The runtime will emulate this feature for [feature level](#) 9.1, 9.2, and 9.3; therefore, this feature is supported for feature level 9.1, 9.2, and 9.3. This feature is always available on new drivers for feature level 10 and higher.

From the shader's point of view, element [0] in the constant buffers array is the constant at *pFirstConstant*.

Out of bounds access to the constant buffers from the shader to the range that is defined by *pFirstConstant* and *pNumConstants* returns 0.

If *pFirstConstant* and *pNumConstants* arrays are **NULL**, you get the same result as if you were binding the entire buffer into view. You get this same result if you call the [DSSetConstantBuffers](#) method. If the buffer is larger than the maximum constant buffer size that is supported by shaders (4096 elements), the shader can access only the first 4096 constants.

If either *pFirstConstant* or *pNumConstants* is **NULL**, the other parameter must also be **NULL**.

# Calling DSSetConstantBuffers1 with command list emulation

The runtime's [command list](#) emulation of **DSSetConstantBuffers1** sometimes doesn't actually change the offsets or sizes for the arrays of constant buffers. This behavior occurs when

**DSSetConstantBuffers1** doesn't effectively change the constant buffers at the beginning and end of the range of slots that you set to update. This section shows how to work around this

behavior.

Here is the code to check whether either the runtime emulates command lists or the driver supports command lists:

C++

```
HRESULT hr = S_OK;
bool needWorkaround = false;
D3D11_DEVICE_CONTEXT_TYPE contextType = pDeviceContext->GetType();

if( D3D11_DEVICE_CONTEXT_DEFERRED == contextType )
{
    D3D11_FEATURE_DATA_THREADING threadingCaps = { FALSE, FALSE };

    hr = pDevice->CheckFeatureSupport( D3D11_FEATURE_THREADING,
&threadingCaps, sizeof(threadingCaps) );
    if( SUCCEEDED(hr) && !threadingCaps.DriverCommandLists )
    {
        needWorkaround = true; // the runtime emulates command lists.
    }
}
```

If the runtime emulates command lists, you need to use one of these code snippets:

If you change the offset and size on only a single constant buffer, set the constant buffer to **NULL** first:

C++

```
pDeviceContext->DSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
if( needWorkaround )
{
    // Workaround for command list emulation
```

```
    pDeviceContext->DSSetConstantBuffers(0, 1, &NullCBuf);  
}  
pDeviceContext->DSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
```

If you change multiple constant buffers, set the first and last constant buffers of the range to **NULL** first:

C++

```
pDeviceContext->DSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);  
if( needWorkaround )  
{  
    // Workaround for command list emulation  
    pDeviceContext->DSSetConstantBuffers(0, 1, &NullCBuf);  
    pDeviceContext->DSSetConstantBuffers(3, 1, &NullCBuf);  
}  
pDeviceContext->DSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);
```

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3D11DeviceContext1::GSGetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Gets the constant buffers that the geometry shader pipeline stage uses.

## Syntax

C++

```
void GSGetConstantBuffers1(
    [in]           UINT      StartSlot,
    [in]           UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers,
    [out, optional] UINT      *pFirstConstant,
    [out, optional] UINT      *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[out, optional] ppConstantBuffers

Array of constant buffer interface pointers to be returned by the method.

[out, optional] pFirstConstant

A pointer to an array that receives the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 2 indicates that the start of the associated constant

buffer is 32 bytes into the constant buffer. The runtime sets *pFirstConstant* to **NULL** if the buffers do not have offsets.

[out, optional] *pNumConstants*

A pointer to an array that receives the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. The runtime sets *pNumConstants* to **NULL** if it doesn't specify the numbers of constants in each buffer.

## Return value

None

## Remarks

If no buffer is bound at a slot, *pFirstConstant* and *pNumConstants* are **NULL** for that slot.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?  Yes  No

Get help at Microsoft Q&A

# ID3D11DeviceContext1::GSSetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Sets the constant buffers that the geometry shader pipeline stage uses.

## Syntax

C++

```
void GSSetConstantBuffers1(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers,
    [in, optional] const UINT   *pFirstConstant,
    [in, optional] const UINT   *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the device's zero-based array to begin setting constant buffers to (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to set (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[in, optional] ppConstantBuffers

Array of constant buffers (see [ID3D11Buffer](#)) being given to the device.

[in, optional] pFirstConstant

An array that holds the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 16 indicates that the start of the associated constant buffer is 256 bytes into the constant buffer. Each offset must be a multiple of 16 constants.

[in, optional] *pNumConstants*

An array that holds the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. Each number of constants must be a multiple of 16 constants, in the range [0..4096].

## Return value

None

## Remarks

The runtime drops the call to **GSSetConstantBuffers1** if the number of constants to which *pNumConstants* points is larger than the maximum constant buffer size that is supported by shaders (4096 constants). The values in the elements of the *pFirstConstant* and *pFirstConstant + pNumConstants* arrays can exceed the length of each buffer; from the shader's point of view, the constant buffer is the intersection of the actual memory allocation for the buffer and the window [value in an element of *pFirstConstant*, value in an element of *pFirstConstant + pNumConstants*]. The runtime also drops the call to **GSSetConstantBuffers1** on existing drivers that don't support this offsetting.

The runtime will emulate this feature for [feature level](#) 9.1, 9.2, and 9.3; therefore, this feature is supported for feature level 9.1, 9.2, and 9.3. This feature is always available on new drivers for feature level 10 and higher.

From the shader's point of view, element [0] in the constant buffers array is the constant at *pFirstConstant*.

Out of bounds access to the constant buffers from the shader to the range that is defined by *pFirstConstant* and *pNumConstants* returns 0.

If *pFirstConstant* and *pNumConstants* arrays are **NULL**, you get the same result as if you were binding the entire buffer into view. You get this same result if you call the [GSSetConstantBuffers](#) method. If the buffer is larger than the maximum constant buffer size that is supported by shaders (4096 elements), the shader can access only the first 4096 constants.

If either *pFirstConstant* or *pNumConstants* is **NULL**, the other parameter must also be **NULL**.

# Calling GSSetConstantBuffers1 with command list emulation

The runtime's [command list](#) emulation of **GSSetConstantBuffers1** sometimes doesn't actually change the offsets or sizes for the arrays of constant buffers. This behavior occurs when

**GSSetConstantBuffers1** doesn't effectively change the constant buffers at the beginning and end of the range of slots that you set to update. This section shows how to work around this

behavior.

Here is the code to check whether either the runtime emulates command lists or the driver supports command lists:

C++

```
HRESULT hr = S_OK;
bool needWorkaround = false;
D3D11_DEVICE_CONTEXT_TYPE contextType = pDeviceContext->GetType();

if( D3D11_DEVICE_CONTEXT_DEFERRED == contextType )
{
    D3D11_FEATURE_DATA_THREADING threadingCaps = { FALSE, FALSE };

    hr = pDevice->CheckFeatureSupport( D3D11_FEATURE_THREADING,
&threadingCaps, sizeof(threadingCaps) );
    if( SUCCEEDED(hr) && !threadingCaps.DriverCommandLists )
    {
        needWorkaround = true; // the runtime emulates command lists.
    }
}
```

If the runtime emulates command lists, you need to use one of these code snippets:

If you change the offset and size on only a single constant buffer, set the constant buffer to **NULL** first:

C++

```
pDeviceContext->GSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
if( needWorkaround )
{
    // Workaround for command list emulation
```

```
    pDeviceContext->GSSetConstantBuffers(0, 1, &NullCBuf);  
}  
pDeviceContext->GSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
```

If you change multiple constant buffers, set the first and last constant buffers of the range to **NULL** first:

C++

```
pDeviceContext->GSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);  
if( needWorkaround )  
{  
    // Workaround for command list emulation  
    pDeviceContext->GSSetConstantBuffers(0, 1, &NullCBuf);  
    pDeviceContext->GSSetConstantBuffers(3, 1, &NullCBuf);  
}  
pDeviceContext->GSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);
```

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3D11DeviceContext1::HSGetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Gets the constant buffers that the [hull-shader stage](#) uses.

## Syntax

C++

```
void HSGetConstantBuffers1(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers,
    [out, optional] UINT      *pFirstConstant,
    [out, optional] UINT      *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[out, optional] ppConstantBuffers

Array of constant buffer interface pointers to be returned by the method.

[out, optional] pFirstConstant

A pointer to an array that receives the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 2 indicates that the start of the associated constant

buffer is 32 bytes into the constant buffer. The runtime sets *pFirstConstant* to **NULL** if the buffers do not have offsets.

[out, optional] *pNumConstants*

A pointer to an array that receives the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. The runtime sets *pNumConstants* to **NULL** if it doesn't specify the numbers of constants in each buffer.

## Return value

None

## Remarks

If no buffer is bound at a slot, *pFirstConstant* and *pNumConstants* are **NULL** for that slot.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?  Yes  No

Get help at Microsoft Q&A

# ID3D11DeviceContext1::HSSetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Sets the constant buffers that the [hull-shader stage](#) of the pipeline uses.

## Syntax

C++

```
void HSSetConstantBuffers1(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers,
    [in, optional] const UINT   *pFirstConstant,
    [in, optional] const UINT   *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the device's zero-based array to begin setting constant buffers to (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to set (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[in, optional] ppConstantBuffers

Array of constant buffers being given to the device.

[in, optional] pFirstConstant

An array that holds the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 16 indicates that the start of the associated constant buffer is 256 bytes into the constant buffer. Each offset must be a multiple of 16 constants.

[in, optional] *pNumConstants*

An array that holds the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. Each number of constants must be a multiple of 16 constants, in the range [0..4096].

## Return value

None

## Remarks

The runtime drops the call to **HSSetConstantBuffers1** if the number of constants to which *pNumConstants* points is larger than the maximum constant buffer size that is supported by shaders (4096 constants). The values in the elements of the *pFirstConstant* and *pFirstConstant + pNumConstants* arrays can exceed the length of each buffer; from the shader's point of view, the constant buffer is the intersection of the actual memory allocation for the buffer and the window [value in an element of *pFirstConstant*, value in an element of *pFirstConstant + pNumConstants*]. The runtime also drops the call to **HSSetConstantBuffers1** on existing drivers that don't support this offsetting.

The runtime will emulate this feature for [feature level](#) 9.1, 9.2, and 9.3; therefore, this feature is supported for feature level 9.1, 9.2, and 9.3. This feature is always available on new drivers for feature level 10 and higher.

From the shader's point of view, element [0] in the constant buffers array is the constant at *pFirstConstant*.

Out of bounds access to the constant buffers from the shader to the range that is defined by *pFirstConstant* and *pNumConstants* returns 0.

If the *pFirstConstant* and *pNumConstants* arrays are **NULL**, you get the same result as if you were binding the entire buffer into view. You get this same result if you call the [HSSetConstantBuffers](#) method. If the buffer is larger than the maximum constant buffer size that is supported by shaders (4096 elements), the shader can access only the first 4096 constants.

If either *pFirstConstant* or *pNumConstants* is **NULL**, the other parameter must also be **NULL**.

# Calling `HSSetConstantBuffers1` with command list emulation

The runtime's [command list](#) emulation of `HSSetConstantBuffers1` sometimes doesn't actually change the offsets or sizes for the arrays of constant buffers. This behavior occurs when

`HSSetConstantBuffers1` doesn't effectively change the constant buffers at the beginning and end of the range of slots that you set to update. This section shows how to work around this

behavior.

Here is the code to check whether either the runtime emulates command lists or the driver supports command lists:

C++

```
HRESULT hr = S_OK;
bool needWorkaround = false;
D3D11_DEVICE_CONTEXT_TYPE contextType = pDeviceContext->GetType();

if( D3D11_DEVICE_CONTEXT_DEFERRED == contextType )
{
    D3D11_FEATURE_DATA_THREADING threadingCaps = { FALSE, FALSE };

    hr = pDevice->CheckFeatureSupport( D3D11_FEATURE_THREADING,
&threadingCaps, sizeof(threadingCaps) );
    if( SUCCEEDED(hr) && !threadingCaps.DriverCommandLists )
    {
        needWorkaround = true; // the runtime emulates command lists.
    }
}
```

If the runtime emulates command lists, you need to use one of these code snippets:

If you change the offset and size on only a single constant buffer, set the constant buffer to **NULL** first:

C++

```
pDeviceContext->HSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
if( needWorkaround )
{
    // Workaround for command list emulation
```

```
    pDeviceContext->HSSetConstantBuffers(0, 1, &NullCBuf);
}
pDeviceContext->HSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
```

If you change multiple constant buffers, set the first and last constant buffers of the range to **NULL** first:

C++

```
pDeviceContext->HSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);
if( needWorkaround )
{
    // Workaround for command list emulation
    pDeviceContext->HSSetConstantBuffers(0, 1, &NullCBuf);
    pDeviceContext->HSSetConstantBuffers(3, 1, &NullCBuf);
}
pDeviceContext->HSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);
```

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?

Yes

No

Get help at Microsoft Q&A

# ID3D11DeviceContext1::PSGetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Gets the constant buffers that the pixel shader pipeline stage uses.

## Syntax

C++

```
void PSGetConstantBuffers1(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers,
    [out, optional] UINT      *pFirstConstant,
    [out, optional] UINT      *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[out, optional] ppConstantBuffers

Array of constant buffer interface pointers to be returned by the method.

[out, optional] pFirstConstant

A pointer to an array that receives the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 2 indicates that the start of the associated constant

buffer is 32 bytes into the constant buffer. The runtime sets *pFirstConstant* to **NULL** if the buffers do not have offsets.

[out, optional] *pNumConstants*

A pointer to an array that receives the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. The runtime sets *pNumConstants* to **NULL** if it doesn't specify the numbers of constants in each buffer.

## Return value

None

## Remarks

If no buffer is bound at a slot, *pFirstConstant* and *pNumConstants* are **NULL** for that slot.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?  

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

# ID3D11DeviceContext1::PSSetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Sets the constant buffers that the pixel shader pipeline stage uses, and enables the shader to access other parts of the buffer.

## Syntax

C++

```
void PSSetConstantBuffers1(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers,
    [in, optional] const UINT   *pFirstConstant,
    [in, optional] const UINT   *pNumConstants
);
```

## Parameters

[in] StartSlot

Type: [UINT](#)

Index into the device's zero-based array to begin setting constant buffers to (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Type: [UINT](#)

Number of buffers to set (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[in, optional] ppConstantBuffers

Type: [ID3D11Buffer\\*](#)

Array of constant buffers being given to the device.

[in, optional] pFirstConstant

Type: **const UINT\***

An array that holds the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components).

Therefore, an offset of 16 indicates that the start of the associated constant buffer is 256 bytes into the constant buffer. Each offset must be a multiple of 16 constants.

[in, optional] *pNumConstants*

Type: **const UINT\***

An array that holds the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. Each number of constants must be a multiple of 16 constants, in the range [0..4096].

## Return value

None

## Remarks

To enable the shader to access other parts of the buffer, call **PSSetConstantBuffers1** instead of **PSSetConstantBuffers**. **PSSetConstantBuffers1** has additional parameters *pFirstConstant* and *pNumConstants*.

The runtime drops the call to **PSSetConstantBuffers1** if the numbers of constants to which *pNumConstants* points is larger than the maximum constant buffer size that is supported by shaders. The maximum constant buffer size that is supported by shaders holds 4096 constants, where each constant has four 32-bit components.

The values in the elements of the *pFirstConstant* and *pFirstConstant + pNumConstants* arrays can exceed the length of each buffer; from the shader's point of view, the constant buffer is the intersection of the actual memory allocation for the buffer and the following window (range):

[value in an element of *pFirstConstant*, value in an element of *pFirstConstant + pNumConstants*]

That is, the window is the range is from (value in an element of *pFirstConstant*) to (value in an element of *pFirstConstant + pNumConstants*).

The runtime also drops the call to **PSSetConstantBuffers1** on existing drivers that do not support this offsetting.

The runtime will emulate this feature for [feature level](#) 9.1, 9.2, and 9.3; therefore, this feature is supported for feature level 9.1, 9.2, and 9.3. This feature is always available on new drivers for feature level 10 and higher.

From the shader's point of view, element [0] in the constant buffers array is the constant at *pFirstConstant*.

Out of bounds access to the constant buffers from the shader to the range that is defined by *pFirstConstant* and *pNumConstants* returns 0.

If *pFirstConstant* and *pNumConstants* arrays are **NULL**, you get the same result as if you were binding the entire buffer into view. You get this same result if you call the **PSSetConstantBuffers** method. If the buffer is larger than the maximum constant buffer size that is supported by shaders (4096 elements), the shader can access only the first 4096 constants.

If either *pFirstConstant* or *pNumConstants* is **NULL**, the other parameter must also be **NULL**.

## Calling **PSSetConstantBuffers1** with command list emulation

The runtime's [command list](#) emulation of **PSSetConstantBuffers1** sometimes doesn't actually change the offsets or sizes for the arrays of constant buffers. This behavior occurs when **PSSetConstantBuffers1** doesn't effectively change the constant buffers at the beginning and end of the range of slots that you set to update. This section shows how to work around this behavior.

Here is the code to check whether either the runtime emulates command lists or the driver supports command lists:

C++

```
HRESULT hr = S_OK;
bool needWorkaround = false;
D3D11_DEVICE_CONTEXT_TYPE contextType = pDeviceContext->GetType();

if( D3D11_DEVICE_CONTEXT_DEFERRED == contextType )
{
    D3D11_FEATURE_DATA_THREADING threadingCaps = { FALSE, FALSE };
```

```
    hr = pDevice->CheckFeatureSupport( D3D11_FEATURE_THREADING,
&threadingCaps, sizeof(threadingCaps) );
    if( SUCCEEDED(hr) && !threadingCaps.DriverCommandLists )
    {
        needWorkaround = true; // the runtime emulates command lists.
    }
}
```

If the runtime emulates command lists, you need to use one of these code snippets:

If you change the offset and size on only a single constant buffer, set the constant buffer to **NULL** first:

C++

```
pDeviceContext->PSSetConstantBuffers1(0, 1, &CBuf, &0ffset, &Count);
if( needWorkaround )
{
    // Workaround for command list emulation
    pDeviceContext->PSSetConstantBuffers(0, 1, &NullCBuf);
}
pDeviceContext->PSSetConstantBuffers1(0, 1, &CBuf, &0ffset, &Count);
```

If you change multiple constant buffers, set the first and last constant buffers of the range to **NULL** first:

C++

```
pDeviceContext->PSSetConstantBuffers1(0, 4, &CBufs, &0ffsets, &Counts);
if( needWorkaround )
{
    // Workaround for command list emulation
    pDeviceContext->PSSetConstantBuffers(0, 1, &NullCBuf);
    pDeviceContext->PSSetConstantBuffers(3, 1, &NullCBuf);
}
pDeviceContext->PSSetConstantBuffers1(0, 4, &CBufs, &0ffsets, &Counts);
```

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::VSGetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Gets the constant buffers that the vertex shader pipeline stage uses.

## Syntax

C++

```
void VSGetConstantBuffers1(
    [in]             UINT      StartSlot,
    [in]             UINT      NumBuffers,
    [out, optional] ID3D11Buffer **ppConstantBuffers,
    [out, optional] UINT      *pFirstConstant,
    [out, optional] UINT      *pNumConstants
);
```

## Parameters

[in] StartSlot

Index into the device's zero-based array to begin retrieving constant buffers from (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Number of buffers to retrieve (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[out, optional] ppConstantBuffers

Array of constant buffer interface pointers to be returned by the method.

[out, optional] pFirstConstant

A pointer to an array that receives the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 2 indicates that the start of the associated constant

buffer is 32 bytes into the constant buffer. The runtime sets *pFirstConstant* to **NULL** if the buffers do not have offsets.

[out, optional] *pNumConstants*

A pointer to an array that receives the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. The runtime sets *pNumConstants* to **NULL** if it doesn't specify the numbers of constants in each buffer.

## Return value

None

## Remarks

If no buffer is bound at a slot, *pFirstConstant* and *pNumConstants* are **NULL** for that slot.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?  

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

# ID3D11DeviceContext1::VSSetConstantBuffers1 method (d3d11\_1.h)

Article 10/13/2021

Sets the constant buffers that the vertex shader pipeline stage uses.

## Syntax

C++

```
void VSSetConstantBuffers1(
    [in]          UINT      StartSlot,
    [in]          UINT      NumBuffers,
    [in, optional] ID3D11Buffer * const *ppConstantBuffers,
    [in, optional] const UINT   *pFirstConstant,
    [in, optional] const UINT   *pNumConstants
);
```

## Parameters

[in] StartSlot

Type: **UINT**

Index into the device's zero-based array to begin setting constant buffers to (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - 1).

[in] NumBuffers

Type: **UINT**

Number of buffers to set (ranges from 0 to D3D11\_COMMONSHADER\_CONSTANT\_BUFFER\_API\_SLOT\_COUNT - *StartSlot*).

[in, optional] ppConstantBuffers

Type: **ID3D11Buffer\***

Array of constant buffers being given to the device.

[in, optional] pFirstConstant

Type: **const UINT\***

An array that holds the offsets into the buffers that *ppConstantBuffers* specifies. Each offset specifies where, from the shader's point of view, each constant buffer starts. Each offset is measured in shader constants, which are 16 bytes (4\*32-bit components). Therefore, an offset of 16 indicates that the start of the associated constant buffer is 256 bytes into the constant buffer. Each offset must be a multiple of 16 constants.

[in, optional] *pNumConstants*

Type: **const UINT\***

An array that holds the numbers of constants in the buffers that *ppConstantBuffers* specifies. Each number specifies the number of constants that are contained in the constant buffer that the shader uses. Each number of constants starts from its respective offset that is specified in the *pFirstConstant* array. Each number of constants must be a multiple of 16 constants, in the range [0..4096].

## Return value

None

## Remarks

The runtime drops the call to **VSSetConstantBuffers1** if the number of constants to which *pNumConstants* points is larger than the maximum constant buffer size that is supported by shaders (4096 constants). The values in the elements of the *pFirstConstant* and *pFirstConstant* + *pNumConstants* arrays can exceed the length of each buffer; from the shader's point of view, the constant buffer is the intersection of the actual memory allocation for the buffer and the window [value in an element of *pFirstConstant*, value in an element of *pFirstConstant* + value in an element of *pNumConstants*]. The runtime also drops the call to **VSSetConstantBuffers1** on existing drivers that don't support this offsetting.

The runtime will emulate this feature for [feature level](#) 9.1, 9.2, and 9.3; therefore, this feature is supported for feature level 9.1, 9.2, and 9.3. This feature is always available on new drivers for feature level 10 and higher.

From the shader's point of view, element [0] in the constant buffers array is the constant at *pFirstConstant*.

Out of bounds access to the constant buffers from the shader to the range that is defined by *pFirstConstant* and *pNumConstants* returns 0.

If *pFirstConstant* and *pNumConstants* arrays are **NULL**, you get the same result as if you were binding the entire buffer into view. You get this same result if you call the [VSSetConstantBuffers](#) method. If the buffer is larger than the maximum constant buffer size that is supported by shaders (4096 elements), the shader can access only the first 4096 constants.

If either *pFirstConstant* or *pNumConstants* is **NULL**, the other parameter must also be **NULL**.

## Calling VSSetConstantBuffers1 with command list emulation

The runtime's [command list](#) emulation of **VSSetConstantBuffers1** sometimes doesn't actually change the offsets or sizes for the arrays of constant buffers. This behavior occurs when

**VSSetConstantBuffers1** doesn't effectively change the constant buffers at the beginning and end of the range of slots that you set to update. This section shows how to work around this

behavior.

Here is the code to check whether either the runtime emulates command lists or the driver supports command lists:

C++

```
HRESULT hr = S_OK;
bool needWorkaround = false;
D3D11_DEVICE_CONTEXT_TYPE contextType = pDeviceContext->GetType();

if( D3D11_DEVICE_CONTEXT_DEFERRED == contextType )
{
    D3D11_FEATURE_DATA_THREADING threadingCaps = { FALSE, FALSE };

    hr = pDevice->CheckFeatureSupport( D3D11_FEATURE_THREADING,
&threadingCaps, sizeof(threadingCaps) );
    if( SUCCEEDED(hr) && !threadingCaps.DriverCommandLists )
    {
        needWorkaround = true; // the runtime emulates command lists.
    }
}
```

If the runtime emulates command lists, you need to use one of these code snippets:

If you change the offset and size on only a single constant buffer, set the constant buffer to **NULL** first:

```
C++  
  
pDeviceContext->VSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);  
if( needWorkaround )  
{  
    // Workaround for command list emulation  
    pDeviceContext->VSSetConstantBuffers(0, 1, &NullCBuf);  
}  
pDeviceContext->VSSetConstantBuffers1(0, 1, &CBuf, &Offset, &Count);
```

If you change multiple constant buffers, set the first and last constant buffers of the range to **NULL** first:

```
C++  
  
pDeviceContext->VSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);  
if( needWorkaround )  
{  
    // Workaround for command list emulation  
    pDeviceContext->VSSetConstantBuffers(0, 1, &NullCBuf);  
    pDeviceContext->VSSetConstantBuffers(3, 1, &NullCBuf);  
}  
pDeviceContext->VSSetConstantBuffers1(0, 4, &CBufs, &Offsets, &Counts);
```

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::SwapDeviceContextState method (d3d11\_1.h)

Article 10/13/2021

Activates the given context state object and changes the current device behavior to Direct3D 11, Direct3D 10.1, or Direct3D 10.

## Syntax

C++

```
void SwapDeviceContextState(  
    [in]           ID3DDeviceContextState *pState,  
    [out, optional] ID3DDeviceContextState **ppPreviousState  
) ;
```

## Parameters

[in] pState

A pointer to the [ID3DDeviceContextState](#) interface for the context state object that was previously created through the [ID3D11Device1::CreateDeviceContextState](#) method. If **SwapDeviceContextState** is called with *pState* set to **NULL**, the call has no effect.

[out, optional] ppPreviousState

A pointer to a variable that receives a pointer to the [ID3DDeviceContextState](#) interface for the previously-activated context state object.

## Return value

None

## Remarks

**SwapDeviceContextState** changes device behavior. This device behavior depends on the emulated interface that you passed to the *EmulatedInterface* parameter of the [ID3D11Device1::CreateDeviceContextState](#) method when you created the context state object.

**SwapDeviceContextState** is not supported on a deferred context.

**SwapDeviceContextState** disables the incompatible device interfaces [ID3D10Device](#), [ID3D10Device1](#), [ID3D11Device](#), and [ID3D11Device1](#). When a context state object is active, the runtime disables certain methods on the device and context interfaces. A context state object that is created with `__uuidof(ID3D11Device1)` or `__uuidof(ID3D11Device)` turns off most of the Direct3D 10 device interfaces. A context state object that is created with `__uuidof(ID3D10Device1)` or `__uuidof(ID3D10Device)` turns off most of the [ID3D11DeviceContext](#) methods. For more information about this behavior, see [ID3D11Device1::CreateDeviceContextState](#).

**SwapDeviceContextState** activates the context state object specified by *pState*. This means that the device behaviors that are associated with the context state object's feature level and compatible interface are activated on the Direct3D device until the next call to **SwapDeviceContextState**. In addition, any state that was saved when this context state object was last active is now reactivated, so that the previous state is replaced.

**SwapDeviceContextState** sets *ppPreviousState* to the most recently activated context state object. The object allows the caller to save and then later restore the previous device state. This behavior is useful in a plug-in architecture such as Direct2D that shares a Direct3D device with its plug-ins. A Direct2D interface can use context state objects to save and restore the application's state.

If the caller did not previously call the [ID3D11Device1::CreateDeviceContextState](#) method to create a previous context state object, **SwapDeviceContextState** sets *ppPreviousState* to the default context state object. In either case, usage of **SwapDeviceContextState** is the same.

The feature level that is specified by the application, and that is chosen by the context state object from the acceptable list that the application supplies to [ID3D11Device1::CreateDeviceContextState](#), controls the feature level of the immediate context whenever the context state object is active. Because the Direct3D 11 device is free-threaded, the device methods cannot query the current immediate context feature level. Instead, the device runs at a feature level that is the maximum of all previously created context state objects' feature levels. This means that the device's feature level can increase dynamically.

The feature level of the context state object controls the functionality available from the immediate context. However, to maintain the free-threaded contract of the Direct3D 11 device methods—the resource-creation methods in particular—the upper-bound feature level of all created context state objects controls the set of resources that the device creates.

Because the context state object interface is published by the immediate context, the interface requires the same threading model as the immediate context. Specifically, **SwapDeviceContextState** is single-threaded with respect to the other immediate context methods and with respect to the equivalent methods of [ID3D10Device](#).

Crucially, because only one of the Direct3D 10 or Direct3D 11 ref-count behaviors can be available at a time, one of the Direct3D 10 and Direct3D 11 interfaces must break its ref-count contract. To avoid this situation, the activation of a context state object turns off the incompatible version interface. Also, if you call a method of an incompatible version interface, the call silently fails if the method has return type **void**, returns an **HRESULT** value of **E\_INVALIDARG**, or sets any out parameter to **NULL**.

When you switch from Direct3D 11 mode to either Direct3D 10 mode or Direct3D 10.1 mode, the binding behavior of the device changes. Specifically, the final release of a resource induces unbind in Direct3D 10 mode or Direct3D 10.1 mode. During final release an application releases all of the resource's references, including indirect references such as the linkage from view to resource, and the linkage from context state object to any of the context state object's bound resources. Any bound resource to which the application has no reference is unbound and destroyed, in order to maintain the Direct3D 10 behavior.

**SwapDeviceContextState** does not affect any state that [ID3D11VideoContext](#) sets.

Command lists that are generated by deferred contexts do not hold a reference to context state objects and are not affected by future updates to context state objects.

No asynchronous objects are affected by **SwapDeviceContextState**. For example, if a query is active before a call to **SwapDeviceContextState**, it is still active after the call.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext1::UpdateSubresource1 method (d3d11\_1.h)

Article 07/27/2022

The CPU copies data from memory to a subresource created in non-mappable memory.

## Syntax

C++

```
void UpdateSubresource1(
    [in]          ID3D11Resource  *pDstResource,
    [in]          UINT           DstSubresource,
    [in, optional] const D3D11_BOX *pDstBox,
    [in]          const void     *pSrcData,
    [in]          UINT           SrcRowPitch,
    [in]          UINT           SrcDepthPitch,
    [in]          UINT           CopyFlags
);
```

## Parameters

[in] pDstResource

Type: [ID3D11Resource\\*](#)

A pointer to the destination resource.

[in] DstSubresource

Type: [UINT](#)

A zero-based index that identifies the destination subresource. See [D3D11CalcSubresource](#) for more details.

[in, optional] pDstBox

Type: [const D3D11\\_BOX\\*](#)

A pointer to a box that defines the portion of the destination subresource to copy the resource data into. Coordinates are in bytes for buffers and in texels for textures. If **NULL**, **UpdateSubresource1** writes the data to the destination subresource with no offset. The dimensions of the source must fit the destination.

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, **UpdateSubresource1** doesn't perform an update operation.

[in] pSrcData

Type: **const void\***

A pointer to the source data in memory.

[in] SrcRowPitch

Type: **UINT**

The size of one row of the source data.

[in] SrcDepthPitch

Type: **UINT**

The size of one depth slice of source data.

[in] CopyFlags

Type: **UINT**

A [D3D11\\_COPY\\_FLAGS](#)-typed value that specifies how to perform the update operation. If you specify zero for no update option, **UpdateSubresource1** behaves like [ID3D11DeviceContext::UpdateSubresource](#). For existing display drivers that can't process these flags, the runtime doesn't use them.

## Return value

None

## Remarks

If you call **UpdateSubresource1** to update a constant buffer, pass any region, and the driver has not been implemented to Windows 8, the runtime drops the call (except [feature level](#) 9.1, 9.2, and 9.3 where the runtime emulates support). The runtime also drops the call if you update a constant buffer with a partial region whose extent is not aligned to 16-byte granularity (16 bytes being a full constant). When the runtime drops the call, the runtime doesn't call the corresponding device driver interface (DDI).

When you record a call to [UpdateSubresource](#) with an offset *pDstBox* in a software command list, the offset in *pDstBox* is incorrectly applied to *pSrcData* when you play back the command list. The new-for-Windows 8 [UpdateSubresource1](#) fixes this issue. In a call to [UpdateSubresource1](#), *pDstBox* does not affect *pSrcData*.

For info about various resource types and how [UpdateSubresource1](#) might work with each resource type, see [Introduction to a Resource in Direct3D 11](#).

**Note** Applies only to feature level 9\_x hardware If you use [UpdateSubresource1](#) or [ID3D11DeviceContext1::CopySubresourceRegion1](#) to copy from a staging resource to a default resource, you can corrupt the destination contents. This occurs if you pass a **NULL** source box and if the source resource has different dimensions from those of the destination resource or if you use destination offsets, (x, y, and z). In this situation, always pass a source box that is the full size of the source resource.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3D11DeviceContext2 interface (d3d11\_2.h)

Article 02/22/2024

The device context interface represents a device context; it is used to render commands. **ID3D11DeviceContext2** adds new methods to those in [ID3D11DeviceContext1](#).

## Inheritance

The **ID3D11DeviceContext2** interface inherits from [ID3D11DeviceContext1](#).

**ID3D11DeviceContext2** also has these types of members:

## Methods

The **ID3D11DeviceContext2** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11DeviceContext2::BeginEventInt</a>
Allows applications to annotate the beginning of a range of graphics commands.
<a href="#">ID3D11DeviceContext2::CopyTileMappings</a>
Copies mappings from a source tiled resource to a destination tiled resource.
<a href="#">ID3D11DeviceContext2::CopyTiles</a>
Copies tiles from buffer to tiled resource or vice versa. ( <code>ID3D11DeviceContext2::CopyTiles</code> )
<a href="#">ID3D11DeviceContext2::EndEvent</a>
Allows applications to annotate the end of a range of graphics commands.
<a href="#">ID3D11DeviceContext2::IsAnnotationEnabled</a>
Allows apps to determine when either a capture or profiling request is enabled.
<a href="#">ID3D11DeviceContext2::ResizeTilePool</a>
Resizes a tile pool.

### [ID3D11DeviceContext2::SetMarkerInt](#)

Allows applications to annotate graphics commands.

### [ID3D11DeviceContext2::TiledResourceBarrier](#)

Specifies a data access ordering constraint between multiple tiled resources.

### [ID3D11DeviceContext2::UpdateTileMappings](#)

Updates mappings of tile locations in tiled resources to memory locations in a tile pool.

### [ID3D11DeviceContext2::UpdateTiles](#)

Updates tiles by copying from app memory to the tiled resource.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h

## See also

[Core Interfaces](#)

[ID3D11DeviceContext1](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::BeginEventInt method (d3d11\_2.h)

Article 02/22/2024

Allows applications to annotate the beginning of a range of graphics commands.

## Syntax

C++

```
void BeginEventInt(  
    [in] LPCWSTR pLabel,  
    INT Data  
)
```

## Parameters

[in] `pLabel`

An optional string that will be logged to [ETW](#) when ETW logging is active. If '#d' appears in the string, it will be replaced by the value of the `Data` parameter similar to the way `printf` works.

`Data`

A signed data value that will be logged to ETW when ETW logging is active.

## Return value

None

## Remarks

`BeginEventInt` allows applications to annotate the beginning of a range of graphics commands, in order to provide more context to what the GPU is executing. When [ETW](#) logging (or a supported tool) is enabled, an additional marker is correlated between the CPU and GPU timelines. The `pLabel` and `Data` value are logged to ETW. When the appropriate ETW logging is not enabled, this method does nothing.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h

## See also

[ID3D11DeviceContext2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::CopyTiles method (d3d11\_2.h)

Article 07/27/2022

Copies tiles from buffer to tiled resource or vice versa.

## Syntax

C++

```
void CopyTiles(  
    [in] ID3D11Resource                 *pTiledResource,  
    [in] const D3D11_TILED_RESOURCE_COORDINATE *pTileRegionStartCoordinate,  
    [in] const D3D11_TILE_REGION_SIZE        *pTileRegionSize,  
    [in] ID3D11Buffer                    *pBuffer,  
    [in] UINT64                          BufferStartOffsetInBytes,  
    [in] UINT                            Flags  
) ;
```

## Parameters

[in] pTiledResource

Type: [ID3D11Resource\\*](#)

A pointer to a tiled resource.

[in] pTileRegionStartCoordinate

Type: [const D3D11\\_TILED\\_RESOURCE\\_COORDINATE\\*](#)

A pointer to a [D3D11\\_TILED\\_RESOURCE\\_COORDINATE](#) structure that describes the starting coordinates of the tiled resource.

[in] pTileRegionSize

Type: [const D3D11\\_TILE\\_REGION\\_SIZE\\*](#)

A pointer to a [D3D11\\_TILE\\_REGION\\_SIZE](#) structure that describes the size of the tiled region.

[in] pBuffer

Type: [ID3D11Buffer](#)\*

A pointer to an [ID3D11Buffer](#) that represents a default, dynamic, or staging buffer.

[in] BufferStartOffsetInBytes

Type: [UINT64](#)

The offset in bytes into the buffer at *pBuffer* to start the operation.

[in] Flags

Type: [UINT](#)

A combination of [D3D11\\_TILE\\_COPY\\_FLAG](#)-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).

If a copy operation involves writing to the same memory location multiple times because multiple locations in the destination resource are mapped to the same tile memory, the resulting write operations to multi-mapped tiles are non-deterministic and non-repeatable; that is, accesses to the tile memory happen in whatever order the hardware happens to execute the copy operation.

The tiles involved in the copy operation can't include tiles that contain packed mipmaps or results of the copy operation are undefined. To transfer data to and from mipmaps that the hardware packs into the one-or-more tiles that constitute the packed mips, you must use the standard (that is, non-tile specific) copy and update APIs (like [ID3D11DeviceContext1::CopySubresourceRegion1](#) and [ID3D11DeviceContext1::UpdateSubresource1](#)) or [ID3D11DeviceContext::GenerateMips](#) for the whole mipmap chain.

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 deswizzle 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](#) or [ID3D11DeviceContext2::UpdateTiles](#) to copy to mipmaps that the hardware packs together. You can just use generic copy and update APIs (like [ID3D11DeviceContext1::CopySubresourceRegion1](#) and [ID3D11DeviceContext1::UpdateSubresource1](#)) to copy small mipmaps individually. Although in the case of a generic copy API (like [ID3D11DeviceContext1::CopySubresourceRegion1](#)), the linear memory must be the same dimension as the tiled resource; [ID3D11DeviceContext1::CopySubresourceRegion1](#) can't copy from a buffer resource to a Texture2D for instance.

For more info about tiled resources, see [Tiled resources](#).

## Requirements

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext2](#)

## Feedback

Was this page helpful?

Yes

No

Get help at Microsoft Q&A

# ID3D11DeviceContext2::CopyTileMappings method (d3d11\_2.h)

Article 10/13/2021

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

## Syntax

C++

```
HRESULT CopyTileMappings(
    [in] ID3D11Resource                  *pDestTiledResource,
    [in] const D3D11_TILED_RESOURCE_COORDINATE *pDestRegionStartCoordinate,
    [in] ID3D11Resource                  *pSourceTiledResource,
    [in] const D3D11_TILED_RESOURCE_COORDINATE *pSourceRegionStartCoordinate,
    [in] const D3D11_TILE_REGION_SIZE        *pTileRegionSize,
    [in] UINT                           Flags
);
```

## Parameters

[in] pDestTiledResource

Type: [ID3D11Resource\\*](#)

A pointer to the destination tiled resource.

[in] pDestRegionStartCoordinate

Type: [const D3D11\\_TILED\\_RESOURCE\\_COORDINATE\\*](#)

A pointer to a [D3D11\\_TILED\\_RESOURCE\\_COORDINATE](#) structure that describes the starting coordinates of the destination tiled resource.

[in] pSourceTiledResource

Type: [ID3D11Resource\\*](#)

A pointer to the source tiled resource.

[in] pSourceRegionStartCoordinate

Type: [const D3D11\\_TILED\\_RESOURCE\\_COORDINATE\\*](#)

A pointer to a [D3D11\\_TILED\\_RESOURCE\\_COORDINATE](#) structure that describes the starting coordinates of the source tiled resource.

[in] pTileRegionSize

Type: [const D3D11\\_TILE\\_REGION\\_SIZE\\*](#)

A pointer to a [D3D11\\_TILE\\_REGION\\_SIZE](#) structure that describes the size of the tiled region.

[in] Flags

Type: [UINT](#)

A combination of [D3D11\\_TILE\\_MAPPING\\_FLAGS](#) values that are combined by using a bitwise OR operation. The only valid value is [D3D11\\_TILE\\_MAPPING\\_NO\\_OVERWRITE](#), which indicates that previously submitted commands to the device that may still be executing do not reference any of the tile region being updated. The device can then avoid having to flush previously submitted work to perform the tile mapping update. If the app violates this promise by updating tile mappings for locations in tiled resources that are still being referenced by outstanding commands, undefined rendering behavior results, including the potential for significant slowdowns on some architectures. This is like the "no overwrite" concept that exists elsewhere in the Direct3D API, except applied to the tile mapping data structure itself (which in hardware is a page table). The absence of the [D3D11\\_TILE\\_MAPPING\\_NO\\_OVERWRITE](#) value requires that tile mapping updates that [CopyTileMappings](#) specifies must be completed before any subsequent Direct3D command can proceed.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the following:

- Returns [E\\_INVALIDARG](#) if various conditions such as invalid flags or passing in non Tiled Resources result in the call being dropped. The dest and the source regions must each entirely fit in their resource or behavior is undefined (debug layer will emit an error).
- Returns [E\\_OUTOFMEMORY](#) if the call results in the driver having to allocate space for new page table mappings but running out of memory. If out of memory occurs when this is called in a commandlist and the commandlist is being executed, the device will be removed. Applications can avoid this situation by only doing update

calls that change existing mappings from Tiled Resources within commandlists (so drivers will not have to allocate page table memory, only change the mapping).

## Remarks

**CopyTileMappings** helps with tasks such as shifting mappings around within and across tiled 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 temp location and then from there written to the destination.

For more info about tiled resources, see [Tiled resources](#).

## Requirements

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::EndEvent method (d3d11\_2.h)

Article 02/22/2024

Allows applications to annotate the end of a range of graphics commands.

## Syntax

C++

```
void EndEvent();
```

## Return value

None

## Remarks

**EndEvent** allows applications to annotate the end of a range of graphics commands, in order to provide more context to what the GPU is executing. When the appropriate [ETW](#) logging is not enabled, this method does nothing. When ETW logging is enabled, an additional marker is correlated between the CPU and GPU timelines.

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::IsAnnotationEnabled method (d3d11\_2.h)

Article02/22/2024

Allows apps to determine when either a capture or profiling request is enabled.

## Syntax

C++

```
BOOL IsAnnotationEnabled();
```

## Return value

Returns **TRUE** if capture or profiling is enabled and **FALSE** otherwise.

## Remarks

Returns **TRUE** if the capture tool is present and capturing or the app is being profiled such that [SetMarkerInt](#) or [BeginEventInt](#) will be logged to [ETW](#). Otherwise, it returns **FALSE**. Apps can use this to turn off self-throttling mechanisms in order to accurately capture what is currently being seen as app output. Apps can also avoid generating event markers and the associated overhead it may entail when there is no benefit to do so.

If apps detect that capture is being performed, they can prevent the Direct3D debugging tools, such as Microsoft Visual Studio 2013, from capturing them. The purpose of the

[D3D11\\_CREATE\\_DEVICE\\_PREVENT\\_ALTERING\\_LAYER\\_SETTINGS\\_FROM\\_REGISTRY](#) flag prior to Windows 8.1 was to allow the Direct3D runtime to prevent debugging tools from capturing apps.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h

## See also

[ID3D11DeviceContext2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::ResizeTilePool method (d3d11\_2.h)

Article 10/13/2021

Resizes a tile pool.

## Syntax

C++

```
HRESULT ResizeTilePool(  
    [in] ID3D11Buffer *pTilePool,  
    [in] UINT64      NewSizeInBytes  
>;
```

## Parameters

[in] pTilePool

Type: [ID3D11Buffer\\*](#)

A pointer to an [ID3D11Buffer](#) for the tile pool to resize.

[in] NewSizeInBytes

Type: [UINT64](#)

The new size in bytes of the tile pool. The size must be a multiple of 64 KB or 0.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the following:

- Returns E\_INVALIDARG if the new tile pool size isn't a multiple of 64 KB or 0.
- Returns E\_OUTOFMEMORY if the call results in the driver having to allocate space for new page table mappings but running out of memory.
- Returns DXGI\_ERROR\_DEVICE\_REMOVED if the video card has been physically removed from the system, or a driver upgrade for the video card has occurred.

For **E\_INVALIDARG** or **E\_OUTOFMEMORY**, the existing tile pool remains unchanged, which includes existing mappings.

## Remarks

**ResizeTilePool** increases or decreases the size of the tile pool depending on whether the app needs more or less working set for the tiled resources that are mapped into it. An app can allocate additional tile pools for new tiled resources, but if any single tiled resource needs more space than initially available in its tile pool, the app can increase the size of the resource's tile pool. A tiled resource can't have mappings into multiple tile pools simultaneously.

When you increase the size of a tile pool, additional tiles are added to the end of the tile pool via one or more new allocations by the driver; your app can't detect the breakdown into the new allocations. Existing memory in the tile pool is left untouched, and existing tiled resource mappings into that memory remain intact.

When you decrease the size of a tile pool, tiles are removed from the end (this is allowed even below the initial allocation size, down to 0). This means that new mappings can't be made past the new size. But, existing mappings past the end of the new size remain intact and useable. The memory is kept active as long as mappings to any part of the allocations that are being used for the tile pool memory remains. If after decreasing, some memory has been kept active because tile mappings are pointing to it and the tile pool is increased again (by any amount), the existing memory is reused first before any additional allocations occur to service the size of the increase.

To be able to save memory, an app has to not only decrease a tile pool but also remove and remap existing mappings past the end of the new smaller tile pool size.

The act of decreasing (and removing mappings) doesn't necessarily produce immediate memory savings. Freeing of memory depends on how granular the driver's underlying allocations for the tile pool are. When a decrease in the size of a tile pool happens to be enough to make a driver allocation unused, the driver can free the allocation. If a tile pool was increased and if you then decrease to previous sizes (and remove and remap tile mappings correspondingly), you will most likely yield memory savings. But, this scenario isn't guaranteed in the case that the sizes don't exactly align with the underlying allocation sizes chosen by the driver.

For more info about tiled resources, see [Tiled resources](#).

## Requirements

---

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::SetMarkerInt method (d3d11\_2.h)

Article 10/13/2021

Allows applications to annotate graphics commands.

## Syntax

C++

```
void SetMarkerInt(
    [in] LPCWSTR pLabel,
    INT          Data
);
```

## Parameters

[in] `pLabel`

An optional string that will be logged to [ETW](#) when ETW logging is active. If '#d' appears in the string, it will be replaced by the value of the *Data* parameter similar to the way `printf` works.

`Data`

A signed data value that will be logged to ETW when ETW logging is active.

## Return value

None

## Remarks

`SetMarkerInt` allows applications to annotate graphics commands, in order to provide more context to what the GPU is executing. When ETW logging or a support tool is enabled, an additional marker is correlated between the CPU and GPU timelines. The *pLabel* and *Data* value are logged to ETW. When the appropriate ETW logging is not enabled, this method does nothing.

# Requirements

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h

## See also

[ID3D11DeviceContext2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::TiledResourceBarrier method (d3d11\_2.h)

Article 10/13/2021

Specifies a data access ordering constraint between multiple tiled resources. For more info about this constraint, see Remarks.

## Syntax

C++

```
void TiledResourceBarrier(
    [in, optional] ID3D11DeviceChild *pTiledResourceOrViewAccessBeforeBarrier,
    [in, optional] ID3D11DeviceChild *pTiledResourceOrViewAccessAfterBarrier
);
```

## Parameters

[in, optional] pTiledResourceOrViewAccessBeforeBarrier

Type: [ID3D11DeviceChild\\*](#)

A pointer to an [ID3D11Resource](#) or [ID3D11View](#) for a resource that was created with the [D3D11\\_RESOURCE\\_MISC\\_TILED](#) flag. Access operations on this object must complete before the access operations on the object that *pTiledResourceOrViewAccessAfterBarrier* specifies.

[in, optional] pTiledResourceOrViewAccessAfterBarrier

Type: [ID3D11DeviceChild\\*](#)

A pointer to an [ID3D11Resource](#) or [ID3D11View](#) for a resource that was created with the [D3D11\\_RESOURCE\\_MISC\\_TILED](#) flag. Access operations on this object must begin after the access operations on the object that *pTiledResourceOrViewAccessBeforeBarrier* specifies.

## Return value

None

## Remarks

Apps can use tiled resources to reuse tiles in different resources. But, a device and driver might not be able to determine whether some memory in a tile pool that was just rendered to is now being used for reading.

For example, an app can render to some tiles in a tile pool with one tiled resource but then read from the same tiles by using a different tiled resource. These tiled-resource operations are different from using one resource and then just switching from writing with [ID3D11RenderTargetView](#) to reading with [ID3D11ShaderResourceView](#). The runtime already tracks and handles these one resource operations using [ID3D11RenderTargetView](#) and [ID3D11ShaderResourceView](#).

When an app transitions from accessing (reading or writing) some location in a tile pool with one resource to accessing the same memory (read or write) via another tiled resource (with mappings to the same memory), the app must call [TiledResourceBarrier](#) after the first use of the resource and before the second. The parameters are the *pTiledResourceOrViewAccessBeforeBarrier* for accesses before the barrier (via rendering, copying), and the *pTiledResourceOrViewAccessAfterBarrier* for accesses after the barrier by using the same tile pool memory. If the resources are identical, the app doesn't need to call [TiledResourceBarrier](#) because this kind of hazard is already tracked and handled.

The barrier call informs the driver that operations issued to the resource before the call must complete before any accesses that occur after the call via a different tiled resource that shares the same memory.

Either or both of the parameters (before or after the barrier) can be **NULL**. **NULL** before the barrier means all tiled resource accesses before the barrier must complete before the resource specified after the barrier can be referenced by the graphics processing unit (GPU). **NULL** after the barrier means that any tiled resources accessed after the barrier can only be executed by the GPU after accesses to the tiled resources before the barrier are finished. Both **NULL** means all previous tiled resource accesses are complete before any subsequent tiled resource access can proceed.

An app can pass a view pointer, a resource, or **NULL** for each parameter. Views are allowed not only for convenience but also to allow the app to scope the barrier effect to a relevant portion of a resource.

For more info about tiled resources, see [Tiled resources](#).

## Requirements

---

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::UpdateTileMappings method (d3d11\_2.h)

Article 10/13/2021

Updates mappings of tile locations in tiled resources to memory locations in a tile pool.

## Syntax

C++

```
HRESULT UpdateTileMappings(
    [in]          ID3D11Resource                  *pTiledResource,
    [in]          UINT                           NumTiledResourceRegions,
    [in, optional] const D3D11_TILED_RESOURCE_COORDINATE
    *pTiledResourceRegionStartCoordinates,
    [in, optional] const D3D11_TILE_REGION_SIZE
    *pTiledResourceRegionSizes,
    [in, optional] ID3D11Buffer                 *pTilePool,
    [in]          UINT                           NumRanges,
    [in, optional] const UINT                   *pRangeFlags,
    [in, optional] const UINT                   *pRangeTileCounts,
    [in]          UINT                           Flags
);
```

## Parameters

[in] pTiledResource

Type: [ID3D11Resource\\*](#)

A pointer to the tiled resource.

[in] NumTiledResourceRegions

Type: [UINT](#)

The number of tiled resource regions.

[in, optional] pTiledResourceRegionStartCoordinates

Type: [const D3D11\\_TILED\\_RESOURCE\\_COORDINATE\\*](#)

An array of **D3D11\_TILED\_RESOURCE\_COORDINATE** structures that describe the starting coordinates of the tiled resource regions. The *NumTiledResourceRegions* parameter specifies the number of **D3D11\_TILED\_RESOURCE\_COORDINATE** structures in the array.

[in, optional] *pTiledResourceRegionSizes*

Type: **const D3D11\_TILE\_REGION\_SIZE\***

An array of **D3D11\_TILE\_REGION\_SIZE** structures that describe the sizes of the tiled resource regions. The *NumTiledResourceRegions* parameter specifies the number of **D3D11\_TILE\_REGION\_SIZE** structures in the array.

[in, optional] *pTilePool*

Type: **ID3D11Buffer\***

A pointer to the tile pool.

[in] *NumRanges*

Type: **UINT**

The number of tile-pool ranges.

[in, optional] *pRangeFlags*

Type: **const UINT\***

An array of **D3D11\_TILE\_RANGE\_FLAG** values that describe each tile-pool range. The *NumRanges* parameter specifies the number of values in the array.

[in, optional] *pTilePoolStartOffsets*

Type: **const UINT\***

An array of offsets into the tile pool. These are 0-based tile offsets, counting in tiles (not bytes).

[in, optional] *pRangeTileCounts*

Type: **const UINT\***

An array of tiles.

An array of values that specify the number of tiles in each tile-pool range. The *NumRanges* parameter specifies the number of values in the array.

[in] Flags

Type: [UINT](#)

A combination of [D3D11\\_TILE\\_MAPPING\\_FLAGS](#) values that are combined by using a bitwise OR operation.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the following:

- Returns E\_INVALIDARG if various conditions such as invalid flags result in the call being dropped. The debug layer will emit an error.
- Returns E\_OUTOFMEMORY if the call results in the driver having to allocate space for new page table mappings but running out of memory. If out of memory occurs when this is called in a commandlist and the commandlist is being executed, the device will be removed. Apps can avoid this situation by only doing update calls that change existing mappings from tiled resources within commandlists (so drivers will not have to allocate page table memory, only change the mapping).
- Returns DXGI\_ERROR\_DEVICE\_REMOVED if the video card has been physically removed from the system, or a driver upgrade for the video card has occurred.

## Remarks

In a single call to [UpdateTileMappings](#), you can map one or more ranges of resource tiles to one or more ranges of tile-pool tiles.

You can organize the parameters of [UpdateTileMappings](#) in these ways to perform an update:

- **Tiled resource whose mappings are updated.** This is a resource that was created with the [D3D11\\_RESOURCE\\_MISC\\_TILED](#) flag. Mappings start off all NULL when a resource is initially created.
- **Set of tile regions on the tiled 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.  
*NumTiledResourceRegions* specifies how many regions there are, *pTiledResourceRegionStartCoordinates* and *pTiledResourceRegionSizes* are each arrays that identify the start location and extend of each region. If *NumTiledResourceRegions* is 1, for convenience either or both of the arrays that

describe the regions can be NULL. NULL for *pTiledResourceRegionStartCoordinates* means the start coordinate is all 0s, and NULL for *pTiledResourceRegionSizes* identifies a default region that is the full set of tiles for the entire tiled resource, including all mipmaps, array slices, or both. If *pTiledResourceRegionStartCoordinates* isn't NULL and *pTiledResourceRegionSizes* 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 *pTiledResourceRegionStartCoordinates* without having to send an array of *pTiledResourceRegionSizes* 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.

- **Tile pool that provides memory where tile mappings can go.** A tiled resource can point to a single tile pool at a time. If a new tile pool is specified (for the first time or different from the last time a tile pool was specified), all existing tile mappings for the tiled resource are cleared and the new set of mappings in the current **UpdateTileMappings** call are applied for the new tile pool. If no tile pool is specified (NULL) or the same tile pool as a previous **UpdateTileMappings** call is provided, the **UpdateTileMappings** call just adds the new mappings to existing ones (overwriting on overlap). If **UpdateTileMappings** only defines NULL mappings, you don't need to specify a tile pool because it is irrelevant. But if you specify a tile pool anyway, it takes the same behavior as previously described when providing a tile pool.
- **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 tile pool (default), a count of tiles in the tiled resource to map to a single tile in a tile pool (sharing the tile), a count of tile mappings in the tiled resource to skip and leave as they are, or a count of tiles in the tile pool 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 tiled 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 tiled resource tile specified has a mapping specified.

*pRangeFlags*, *pTilePoolStartOffsets*, and *pRangeTileCounts* are all arrays, of size *NumRanges*, that describe the tile ranges. If *pRangeFlags* is NULL, all ranges are sequential tiles in the tile pool; otherwise, for each range i, *pRangeFlags[i]* identifies how the mappings in that range of tiles work:

- If `pRangeFlags[i]` is 0, that range defines sequential tiles in the tile pool, with the number of tiles being `pRangeTileCounts[i]` and the starting location `pTilePoolStartOffsets[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 `D3D11_TILE_RANGE_REUSE_SINGLE_TILE`, `pTilePoolStartOffsets[i]` identifies the single tile in the tile pool to map to, and `pRangeTileCounts[i]` specifies how many tiles from the tile regions to map to that tile pool 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 `D3D11_TILE_RANGE_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. `pTilePoolStartOffsets[i]` is ignored for NULL mappings.
- If `pRangeFlags[i]` is `D3D11_TILE_RANGE_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. `pTilePoolStartOffsets[i]` is ignored for SKIP mappings.
- **Flags parameter for overall options.** `D3D11_TILE_MAPPING_NO_OVERWRITE` means the caller promises that previously submitted commands to the device that may still be executing do not reference any of the tile region being updated. This allows the device to avoid having to flush previously submitted work in order to do the tile mapping update. If the app violates this promise by updating tile mappings for locations in tiled resources still being referenced by outstanding commands, undefined rendering behavior results, which includes the potential for significant slowdowns on some architectures. This is like the "no overwrite" concept that exists elsewhere in the Direct3D API, except applied to tile mapping data structure itself, which in hardware is a page table. The absence of this flag requires that tile mapping updates specified by this `UpdateTileMappings` call must be completed before any subsequent Direct3D command can proceed.

If tile mappings have changed on a tiled resource that the app will render via `RenderTargetView` or `DepthStencilView`, the app must clear, by using the fixed function `Clear` APIs, the tiles that have changed within the area being rendered (mapped or not). If an app doesn't clear in these situations, the app receives undefined values when it reads from the tiled resource.

**Note** In Direct3D 11.2, hardware can now support `ClearView` on depth-only formats. For more info, see `D3D11_FEATURE_DATA_D3D11_OPTIONS1`.

If an app needs to preserve existing memory contents of areas in a tiled resource where mappings have changed, the app can first save the contents where tile mappings have changed, by copying them to a temporary surface, for example using [CopyTiles](#), issuing the required [Clear](#), and then copying the contents back.

Suppose a tile is mapped into multiple tiled resources at the same time and tile contents are manipulated by any means (render, copy, and so on) via one of the tiled resources. Then, if the same tile is to be rendered via any other tiled resource, the tile must be cleared first as previously described.

For more info about tiled resources, see [Tiled resources](#).

Here are some examples of common [UpdateTileMappings](#) cases:

## Examples

Clearing an entire surface's mappings to NULL:

C++

```
// - No-overwrite is specified, assuming it is known nothing else the GPU
// could be doing is referencing the previous mappings
// - NULL for pTiledResourceRegionStatCoordinates and
// pTiledResourceRegionSizes defaults to the entire resource
// - NULL for pTilePoolStartOffsets 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 tiled resource region (which is
// the entire surface in this case)
//
// UINT RangeFlags = D3D11_TILE_MAPPING_NULL;
// pDeviceContext2-
>UpdateTileMappings(pTiledResource,1,NULL,NULL,NULL,1,&RangeFlags,NULL,NULL,
D3D11_TILE_MAPPING_NO_OVERWRITE);
```

Mapping a region of tiles to a single tile:

C++

```
// - This maps a 2x3 tile region at tile offset (1,1) in a Tiled Resource to
// tile [12] in a Tile Pool
//
// D3D11_TILED_RESOURCE_COORDINATE TRC;
// TRC.X = 1;
// TRC.Y = 1;
```

```

// TRC.Z = 0;
// TRC.Subresource = 0;
//
// D3D11_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 = D3D11_TILE_MAPPING_REUSE_SINGLE_TILE;
// UINT StartOffset = 12;
// pDeviceContext2-
>UpdateTileMappings(pTiledResource,1,&TRC,&TRS,pTilePool,1,&RangeFlags,&Star
tOffset,
//
NULL,D3D11_TILE_MAPPING_NO_OVERWRITE);

```

Defining mappings for a set of disjoint individual tiles:

C++

```

// - This can also be accomplished in multiple calls. Using a single call
to define multiple
// 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 pTiledResourceRegionSizes defaults to each region in
the Tiled Resource
// being a single tile. So all that is needed are the coordinates of each
one.
// - Passing NULL for Range Flags defaults to no flags (since none are
needed in this case)
// - Passing NULL for pRangeTileCounts defaults to each range in the Tile
Pool being size 1.
// So all that is needed are the start offsets for each tile in the Tile
Pool
//
// D3D11_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;
//
// pDeviceContext2-
>UpdateTileMappings(pTiledResource,NumSingleTiles,&TRC,NULL,pTilePool,NumSin
gleTiles,NULL,StartOffsets,
//
NULL,D3D11_TILE_MAPPING_NO_OVERWRITE);

```

Complex example - defining mappings for regions with some skips, some NULL mappings:

C++

```

// - This complex example hard codes the parameter arrays, whereas in
// practice the
// application would likely configure the parameters programatically or in
// a data driven way.
// - Suppose we have 3 regions in a Tiled 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 Tile Pool starting at
// tile pool 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 (tile pool location [17]) and the
// remaining
// 4 tiles to each map to to unique tile pool locations, [2], [9], [4] and
// [17]:
//
// D3D11_TILED_RESOURCE_COORDINATE TRC[3];
// D3D11_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 TilePoolStartOffsets[8];
// UINT RangeTileCounts[8];
//
// RangeFlags[0] = 0;
// TilePoolStartOffsets[0] = 9;
// RangeTileCounts[0] = 3;
//
// RangeFlags[1] = D3D11_TILE_MAPPING_SKIP;
// TilePoolStartOffsets[1] = 0; // offset is ignored for skip mappings
// RangeTileCounts[1] = 8;
//
// RangeFlags[2] = D3D11_TILE_MAPPING_NULL;
// TilePoolStartOffsets[2] = 0; // offset is ignored for NULL mappings
// RangeTileCounts[2] = 2;
//
// RangeFlags[3] = D3D11_TILE_MAPPING_REUSE_SINGLE_TILE;
// TilePoolStartOffsets[3] = 17;
// RangeTileCounts[3] = 5;
//
// RangeFlags[4] = 0;
// TilePoolStartOffsets[4] = 2;
// RangeTileCounts[4] = 1;
//
// RangeFlags[5] = 0;
// TilePoolStartOffsets[5] = 9;
// RangeTileCounts[5] = 1;
//
// RangeFlags[6] = 0;
// TilePoolStartOffsets[6] = 4;
// RangeTileCounts[6] = 1;
//
// RangeFlags[7] = 0;
// TilePoolStartOffsets[7] = 17;
// RangeTileCounts[7] = 1;
//
// pDeviceContext2-
>UpdateTileMappings(pTiledResource,NumRegions,TRC,TRS,pTilePool,NumRanges,Ra
ngeFlags,
//
```

```
TilePoolStartOffsets, RangeTileCounts, D3D11_TILE_MAPPING_NO_OVERWRITE);
```

## CopyTileMappings

C++

```
// CopyTileMappings helps with tasks such as shifting mappings around
// within/across Tiled Resources, e.g. scrolling tiles.
// The source and dest region can overlap - the result of the copy in this
// case is as if the source was saved to a temp and then
// from there written to the dest, though the implementation may be able to
// do better.
//
// The Flags field allows D3D11_TILE_MAPPING_NO_OVERWRITE to be specified,
// means the caller promises that previously
// submitted commands to the device that may still be executing do not
// reference any of the tile region being updated.
// This allows the device to avoid having to flush previously submitted
// work in order to do the tile mapping
// update. If the application violates this promise by updating tile
// mappings for locations in Tiled Resources
// still being referenced by outstanding commands, undefined rendering
// behavior results, including the potential
// for significant slowdowns on some architectures. This is like the
// "no overwrite" concept that exists
// elsewhere in the API, except applied to Tile Mapping data structure
// itself (which in hardware is a page table).
// The absence of this flag requires that tile mapping updates
// specified by this call must be completed before any
// subsequent D3D command can proceed.
//
// Return Values:
//
// Returns S_OK or E_INVALIDARG or E_OUTOFMEMORY. The latter can happen if
// the call results in the driver having to
// allocate space for new page table mappings but running out of memory.
//
// If out of memory occurs when this is called in a commandlist and the
// commandlist is being executed, the device will be removed.
// Applications can avoid this situation by only doing update calls that
// change existing mappings from Tiled Resources
// within commandlists (so drivers will not have to allocate page table
// memory, only change the mapping).
//
// Various other basic conditions such as invalid flags or passing in non
// Tiled Resources result in call being dropped
// with E_INVALIDARG.
//
// Validation remarks:
//
// The dest and the source regions must each entirely fit in their resource
// or behavior is undefined
```

```
// (debug layer will emit an error).
```

# Requirements

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext2::UpdateTiles method (d3d11\_2.h)

Article 10/13/2021

Updates tiles by copying from app memory to the tiled resource.

## Syntax

C++

```
void UpdateTiles(
    [in] ID3D11Resource                  *pDestTiledResource,
    [in] const D3D11_TILED_RESOURCE_COORDINATE
    *pDestTileRegionStartCoordinate,
    [in] const D3D11_TILE_REGION_SIZE      *pDestTileRegionSize,
    [in] const void                      *pSourceTileData,
    [in] UINT                           Flags
);
```

## Parameters

[in] pDestTiledResource

Type: [ID3D11Resource\\*](#)

A pointer to a tiled resource to update.

[in] pDestTileRegionStartCoordinate

Type: [const D3D11\\_TILED\\_RESOURCE\\_COORDINATE\\*](#)

A pointer to a [D3D11\\_TILED\\_RESOURCE\\_COORDINATE](#) structure that describes the starting coordinates of the tiled resource.

[in] pDestTileRegionSize

Type: [const D3D11\\_TILE\\_REGION\\_SIZE\\*](#)

A pointer to a [D3D11\\_TILE\\_REGION\\_SIZE](#) structure that describes the size of the tiled region.

[in] pSourceTileData

Type: **const void\***

A pointer to memory that contains the source tile data that **UpdateTiles** uses to update the tiled resource.

[in] Flags

Type: **UINT**

A combination of [D3D11\\_TILE\\_COPY\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The only valid value is [D3D11\\_TILE\\_COPY\\_NO\\_OVERWRITE](#). The other values aren't meaningful here, though by definition the [D3D11\\_TILE\\_COPY\\_LINEAR\\_BUFFER\\_TO\\_SWIZZLED\\_TILED\\_RESOURCE](#) value is basically what **UpdateTiles** does, but sources from app memory.

## Return value

None

## Remarks

**UpdateTiles** drops write operations to unmapped areas (except on [Tier\\_1](#) tiled resources, where writing to unmapped areas is invalid).

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 and update APIs (like [ID3D11DeviceContext1::CopySubresourceRegion1](#) and [ID3D11DeviceContext1::UpdateSubresource1](#)) or [ID3D11DeviceContext::GenerateMips](#) for the whole mipmap chain.

The memory layout of the data on the source side of the copy operation is linear in memory within 64 KB tiles, which the hardware and driver swizzle and deswizzle 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 [ID3D11DeviceContext2::CopyTiles](#) or [UpdateTiles](#) to copy to mipmaps that the hardware packs together. You can just use generic copy and update APIs (like [ID3D11DeviceContext1::CopySubresourceRegion1](#) and [ID3D11DeviceContext1::UpdateSubresource1](#)) to copy small mipmaps individually. Although in the case of a generic copy API (like [ID3D11DeviceContext1::CopySubresourceRegion1](#)), the linear memory must be the same dimension as the tiled resource; [ID3D11DeviceContext1::CopySubresourceRegion1](#) can't copy from a buffer resource to a Texture2D for instance.

For more info about tiled resources, see [Tiled resources](#).

## Requirements

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_2.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext2](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext3 interface (d3d11\_3.h)

Article 02/22/2024

The device context interface represents a device context; it is used to render commands. **ID3D11DeviceContext3** adds new methods to those in [ID3D11DeviceContext2](#).

## Inheritance

The **ID3D11DeviceContext3** interface inherits from [ID3D11DeviceContext2](#).

**ID3D11DeviceContext3** also has these types of members:

## Methods

The **ID3D11DeviceContext3** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11DeviceContext3::Flush1</a>
Sends queued-up commands in the command buffer to the graphics processing unit (GPU), with a specified context type and an optional event handle to create an event query.
<a href="#">ID3D11DeviceContext3::GetHardwareProtectionState</a>
Gets whether hardware protection is enabled.
<a href="#">ID3D11DeviceContext3::SetHardwareProtectionState</a>
Sets the hardware protection state.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d11_3.h

## See also

[Core Interfaces](#)

[ID3D11DeviceContext](#)

[ID3D11DeviceContext1](#)

[ID3D11DeviceContext2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext3::Flush1 method (d3d11\_3.h)

Article 02/22/2024

Sends queued-up commands in the command buffer to the graphics processing unit (GPU), with a specified context type and an optional event handle to create an event query.

## Syntax

C++

```
void Flush1(
    D3D11_CONTEXT_TYPE ContextType,
    [in, optional] HANDLE hEvent
);
```

## Parameters

ContextType

Type: [D3D11\\_CONTEXT\\_TYPE](#)

A [D3D11\\_CONTEXT\\_TYPE](#) that specifies the context in which a query occurs, such as a 3D command queue, 3D compute queue, 3D copy queue, video, or image.

[in, optional] hEvent

Type: [HANDLE](#)

An optional event handle. When specified, this method creates an event query.

**Flush1** operates asynchronously, therefore it can return either before or after the GPU finishes executing the queued graphics commands, which will eventually complete. To create an event query, you can call [ID3D11Device::CreateQuery](#) with the value [D3D11\\_QUERY\\_EVENT](#) value. To determine when the GPU is finished processing the graphics commands, you can then use that event query in a call to [ID3D11DeviceContext::GetData](#).

## Return value

None

## Remarks

`Flush1` has parameters. For more information, see [ID3D11DeviceContext::Flush](#), which doesn't have parameters.

## Requirements

  [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext3](#)

[ID3D11DeviceContext::Flush](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext3::GetHardwareProtectionState method (d3d11\_3.h)

Article 02/22/2024

Gets whether hardware protection is enabled.

## Syntax

C++

```
void GetHardwareProtectionState(  
    [out] BOOL *pHwProtectionEnable  
);
```

## Parameters

[out] pHwProtectionEnable

Type: **BOOL\***

After this method returns, points to a **BOOL** that indicates whether hardware protection is enabled.

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext3::SetHardwareProtectionState method (d3d11\_3.h)

Article 02/22/2024

Sets the hardware protection state.

## Syntax

C++

```
void SetHardwareProtectionState(  
    [in] BOOL HwProtectionEnable  
);
```

## Parameters

[in] HwProtectionEnable

Type: **BOOL**

Specifies whether to enable hardware protection.

## Return value

None

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext4 interface (d3d11\_3.h)

Article 02/22/2024

The device context interface represents a device context; it is used to render commands. **ID3D11DeviceContext4** adds new methods to those in [ID3D11DeviceContext3](#).

**Note** This interface, introduced in the Windows 10 Creators Update, is the latest version of the [ID3D11DeviceContext](#) interface. Applications targeting Windows 10 Creators Update should use this interface instead of earlier versions.

## Inheritance

The **ID3D11DeviceContext4** interface inherits from [ID3D11DeviceContext3](#). **ID3D11DeviceContext4** also has these types of members:

## Methods

The **ID3D11DeviceContext4** interface has these methods.

[+] Expand table

<a href="#">ID3D11DeviceContext4::Signal</a>	Updates a fence to a specified value after all previous work has completed.
<a href="#">ID3D11DeviceContext4::Wait</a>	Waits until the specified fence reaches or exceeds the specified value before future work can begin.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h

## See also

[Core Interfaces](#)

[ID3D11DeviceContext2](#)

[ID3D11DeviceContext](#)

[ID3D11DeviceContext1](#)

[ID3D11DeviceContext3](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext4::Signal method (d3d11\_3.h)

Article 02/22/2024

Updates a fence to a specified value after all previous work has completed.

This member function is equivalent to the Direct3D 12 [ID3D12CommandQueue::Signal](#) member function, and applies between Direct3D 11 and Direct3D 12 in interop scenarios.

**Note** This method only applies to immediate-mode contexts.

## Syntax

C++

```
HRESULT Signal(  
    ID3D11Fence *pFence,  
    UINT64      Value  
);
```

## Parameters

`pFence`

Type: [ID3D11Fence\\*](#)

A pointer to the [ID3D11Fence](#) object.

`Value`

Type: [UINT64](#)

The value to set the fence to.

## Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

# Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext4](#)

[Multi-engine synchronization](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DeviceContext4::Wait method (d3d11\_3.h)

Article 02/22/2024

Waits until the specified fence reaches or exceeds the specified value before future work can begin.

This member function is equivalent to the Direct3D 12 [ID3D12CommandQueue::Wait](#) member function, and applies between Direct3D 11 and Direct3D 12 in interop scenarios.

**Note** This method only applies to immediate-mode contexts.

## Syntax

C++

```
HRESULT Wait(  
    ID3D11Fence *pFence,  
    UINT64      Value  
) ;
```

## Parameters

pFence

Type: [ID3D11Fence\\*](#)

A pointer to the [ID3D11Fence](#) object.

Value

Type: [UINT64](#)

The value that the device context is waiting for the fence to reach or exceed. So when [ID3D11Fence::GetCompletedValue](#) is greater than or equal to *Value*, the wait is terminated.

# Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Requirements

  [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11DeviceContext4](#)

[Multi-engine synchronization](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3DDeviceContextState interface (d3d11\_1.h)

Article 02/22/2024

The **ID3DDeviceContextState** interface represents a context state object, which holds state and behavior information about a Microsoft Direct3D device.

## Inheritance

The **ID3DDeviceContextState** interface inherits from the **ID3D11DeviceChild** interface.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h

## See also

[Core Interfaces](#)

[ID3D11Device1::CreateDeviceContextState](#)

[ID3D11DeviceChild](#)

[ID3D11DeviceContext1::SwapDeviceContextState](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11Fence interface (d3d11\_3.h)

Article 02/22/2024

Represents a fence, an object used for synchronization of the CPU and one or more GPUs.

This interface is equivalent to the Direct3D 12 [ID3D12Fence](#) interface, and is also used for synchronization between Direct3D 11 and Direct3D 12 in interop scenarios.

## Inheritance

The **ID3D11Fence** interface inherits from [ID3D11DeviceChild](#). **ID3D11Fence** also has these types of members:

## Methods

The **ID3D11Fence** interface has these methods.

[+] Expand table

<a href="#">ID3D11Fence::CreateSharedHandle</a>
Creates a shared handle to a fence object.
<a href="#">ID3D11Fence::GetCompletedValue</a>
Gets the current value of the fence. ( <code>ID3D11Fence.GetCompletedValue</code> )
<a href="#">ID3D11Fence::SetEventOnCompletion</a>
Specifies an event that should be fired when the fence reaches a certain value. ( <code>ID3D11Fence.SetEventOnCompletion</code> )

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d11_3.h

## See also

[Core Interfaces](#)

[Fence Based Resource Management \(Direct3D 12\)](#)

[ID3D11DeviceChild](#)

[Multi-engine synchronization](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Fence::CreateSharedHandle method (d3d11\_3.h)

Article 10/13/2021

Creates a shared handle to a fence object.

This method is equivalent to the Direct3D 12 [ID3D12Device::CreateSharedHandle](#) method, and it applies in scenarios involving interoperation between Direct3D 11 and Direct3D 12. In DirectX 11, you can open the shared fence handle with the [ID3D11Device5::OpenSharedFence](#) method. In DirectX 12, you can open the shared fence handle with the [ID3D12Device::OpenSharedHandle](#) method.

## Syntax

C++

```
HRESULT CreateSharedHandle(
    [in, optional] const SECURITY_ATTRIBUTES *pAttributes,
                           DWORD                  dwAccess,
    [in, optional] LPCWSTR          lpName,
    [out]           HANDLE        *pHandle
);
```

## Parameters

[in, optional] pAttributes

Type: [const SECURITY\\_ATTRIBUTES\\*](#)

A pointer to a [SECURITY\\_ATTRIBUTES](#) structure that contains two separate but related data members: an optional security descriptor, and a **Boolean** value that determines whether child processes can inherit the returned handle.

Set this parameter to **NULL** if you want child processes that the application might create to not inherit the handle returned by [CreateSharedHandle](#), and if you want the resource that is associated with the returned handle to get a default security descriptor.

The **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](#).

`dwAccess`

Type: [DWORD](#)

Currently the only value this parameter accepts is `GENERIC_ALL`.

`[in, optional] lpName`

Type: [LPCWSTR](#)

A NULL-terminated **UNICODE** string that contains the name to associate with the shared heap. The name is limited to `MAX_PATH` characters. Name comparison is case-sensitive.

If *Name* matches the name of an existing resource, [CreateSharedHandle](#) fails with `DXGI_ERROR_NAME_ALREADY_EXISTS`. This occurs because these objects share the same namespace.

The name can have a "Global" or "Local" prefix to explicitly create the object in the global or session namespace. The remainder of the name can contain any character except the backslash character (\). For more information, see [Kernel Object Namespaces](#). Fast user switching is implemented using Terminal Services sessions. Kernel object names must follow the guidelines outlined for Terminal Services so that applications can support multiple users.

The object can be created in a private namespace. For more information, see [Object Namespaces](#).

`[out] pHANDLE`

Type: [HANDLE\\*](#)

A pointer to a variable that receives the NT HANDLE value to the resource to share. You can use this handle in calls to access the resource.

## Return value

Type: [HRESULT](#)

Returns `S_OK` if successful; otherwise, returns one of the following values:

- [DXGI\\_ERROR\\_INVALID\\_CALL](#) if one of the parameters is invalid.

- `DXGI_ERROR_NAME_ALREADY_EXISTS` if the supplied name of the resource to share is already associated with another resource.
- `E_ACCESSDENIED` if the object is being created in a protected namespace.
- `E_OUTOFMEMORY` if sufficient memory is not available to create the handle.
- Possibly other error codes that are described in the [Direct3D 11 Return Codes](#) topic.

## Remarks

In order to create a shared handle for the specified fence, the fence must have been created with either the `D3D11_FENCE_FLAG_SHARED` or `D3D11_FENCE_FLAG_SHARED_CROSS_ADAPTER` flags. For more information see the [D3D11\\_FENCE\\_FLAG](#) enumeration.

## Requirements

Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib
DLL	D3D11.dll

## See also

[ID3D11Fence](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Fence::GetCompletedValue method (d3d11\_3.h)

Article02/22/2024

Gets the current value of the fence.

This member function is equivalent to the Direct3D 12 [ID3D12Fence::GetCompletedValue](#) member function, and applies between Direct3D 11 and Direct3D 12 in interop scenarios.

## Syntax

C++

```
UINT64 GetCompletedValue();
```

## Return value

Type: [UINT64](#)

Returns the current value of the fence.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib
DLL	D3D11.dll

## See also

[ID3D11Fence](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Fence::SetEventOnCompletion method (d3d11\_3.h)

Article 02/22/2024

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

This member function is equivalent to the Direct3D 12 [ID3D12Fence::SetEventOnCompletion](#) member function, and applies between Direct3D 11 and Direct3D 12 in interop scenarios.

## Syntax

C++

```
HRESULT SetEventOnCompletion(
    UINT64 Value,
    HANDLE hEvent
);
```

## Parameters

Value

Type: [UINT64](#)

The fence value when the event is to be signaled.

hEvent

Type: [HANDLE](#)

A handle to the event object.

## Return value

Type: [HRESULT](#)

This method returns [E\\_OUTOFMEMORY](#) if the kernel components don't have sufficient memory to store the event in a list. See [Direct3D 11 Return Codes](#) for other possible return values.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib
DLL	D3D11.dll

## See also

[ID3D11Fence](#)

[Multi-engine synchronization](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InputLayout interface (d3d11.h)

Article 02/22/2024

An input-layout interface holds a definition of how to feed vertex data that is laid out in memory into the [input-assembler stage](#) of the [graphics pipeline](#).

## Inheritance

The **ID3D11InputLayout** interface inherits from the **ID3D11DeviceChild** interface.

## Remarks

To create an input-layout object, call [ID3D11Device::CreateInputLayout](#). To bind the input-layout object to the [input-assembler stage](#), call [ID3D11DeviceContext::IASetInputLayout](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Multithread interface (d3d11\_4.h)

Article 02/22/2024

Provides threading protection for critical sections of a multi-threaded application.

## Inheritance

The **ID3D11Multithread** interface inherits from the [IUnknown](#) interface.

**ID3D11Multithread** also has these types of members:

## Methods

The **ID3D11Multithread** interface has these methods.

[+] [Expand table](#)

<b>ID3D11Multithread::Enter</b>  Enter a device's critical section. ( <code>ID3D11Multithread.Enter</code> )
<b>ID3D11Multithread::GetMultithreadProtected</b>  Find out if multithread protection is turned on or not.
<b>ID3D11Multithread::Leave</b>  Leave a device's critical section. ( <code>ID3D11Multithread.Leave</code> )
<b>ID3D11Multithread::SetMultithreadProtected</b>  Turns multithread protection on or off.

## Remarks

This interface is obtained by querying it from an immediate device context created with the [ID3D11DeviceContext](#) (or later versions of this) interface using [IUnknown::QueryInterface](#).

Unlike D3D10, there is no multithreaded layer in D3D11. By default, multithread protection is turned off. Use [SetMultithreadProtected](#) to turn it on, then [Enter](#) and [Leave](#)

to encapsulate graphics commands that must be executed in a specific order.

By default in D3D11, applications can only use one thread with the immediate context at a time. But, applications can use this interface to change that restriction. The interface can turn on threading protection for the immediate context, which will increase the overhead of each immediate context call in order to share one context with multiple threads.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_4.h

## See also

[Core Interfaces](#)

[IUnknown](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Multithread::Enter method (d3d11\_4.h)

Article 02/22/2024

Enter a device's critical section.

## Syntax

C++

```
void Enter();
```

## Return value

None

## Remarks

If [SetMultithreadProtected](#) is set to true, then entering a device's critical section prevents other threads from simultaneously calling that device's methods, calling DXGI methods, and calling the methods of all resource, view, shader, state, and asynchronous interfaces.

This function should be used in multithreaded applications when there is a series of graphics commands that must happen in order. This function is typically called at the beginning of the series of graphics commands, and [Leave](#) is typically called after those graphics commands.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11_4.h
Library	D3d11_4.lib

Requirement	Value
DLL	D3d11_4.dll

## See also

[ID3D11Multithread](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Multithread::GetMultithreadProtected method (d3d11\_4.h)

Article 02/22/2024

Find out if multithread protection is turned on or not.

## Syntax

C++

```
BOOL GetMultithreadProtected();
```

## Return value

Type: **BOOL**

Returns true if multithread protection is turned on, false otherwise.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11_4.h
Library	D3d11_4.lib
DLL	D3d11_4.dll

## See also

[ID3D11Multithread](#)

[SetMultithreadProtected](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Multithread::Leave method (d3d11\_4.h)

Article 02/22/2024

Leave a device's critical section.

## Syntax

C++

```
void Leave();
```

## Return value

None

## Remarks

This function is typically used in multithreaded applications when there is a series of graphics commands that must happen in order. [Enter](#) is typically called at the beginning of a series of graphics commands, and this function is typically called after those graphics commands.

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11_4.h
Library	D3d11_4.lib
DLL	D3d11_4.dll

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Multithread::SetMultithreadProtected method (d3d11\_4.h)

Article 02/22/2024

Turns multithread protection on or off.

## Syntax

C++

```
BOOL SetMultithreadProtected(  
    [in] BOOL bMTPProtect  
);
```

## Parameters

[in] bMTPProtect

Type: **BOOL**

Set to true to turn multithread protection on, false to turn it off.

## Return value

Type: **BOOL**

True if multithread protection was already turned on prior to calling this method, false otherwise.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11_4.h
Library	D3d11_4.lib

Requirement	Value
DLL	D3d11_4.dll

## See also

[GetMultithreadProtected](#)

[ID3D11Multithread](#)

---

## Feedback

Was this page helpful?



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

# ID3D11Predicate interface (d3d11.h)

Article02/22/2024

A predicate interface determines whether geometry should be processed depending on the results of a previous draw call.

## Inheritance

The **ID3D11Predicate** interface inherits from the **ID3D11Query** interface.

## Remarks

To create a predicate object, call [ID3D11Device::CreatePredicate](#). To set the predicate object, call [ID3D11DeviceContext::SetPredication](#).

There are two types of predicates: stream-output-overflow predicates and occlusion predicates. Stream-output-overflow predicates cause any geometry residing in stream-output buffers that were overflowed to not be processed. Occlusion predicates cause any geometry that did not have a single sample pass the depth/stencil tests to not be processed.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11Query](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Query interface (d3d11.h)

Article 02/22/2024

A query interface queries information from the GPU.

## Inheritance

The **ID3D11Query** interface inherits from [ID3D11Asynchronous](#). **ID3D11Query** also has these types of members:

## Methods

The **ID3D11Query** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11Query::GetDesc</a>
Get a query description. ( <code>ID3D11Query.GetDesc</code> )

## Remarks

A query can be created with [ID3D11Device::CreateQuery](#).

Query data is typically gathered by issuing an [ID3D11DeviceContext::Begin](#) command, issuing some graphics commands, issuing an [ID3D11DeviceContext::End](#) command, and then calling [ID3D11DeviceContext::GetData](#) to get data about what happened in between the Begin and End calls. The data returned by **GetData** will be different depending on the type of query.

There are, however, some queries that do not require calls to **Begin**. For a list of possible queries see [D3D11\\_QUERY](#).

A query is typically executed as shown in the following code:

```
D3D11_QUERY_DESC queryDesc;
```

```

... // Fill out queryDesc structure
ID3D11Query * pQuery;
pDevice->CreateQuery(&queryDesc, &pQuery);
pDeviceContext->Begin(pQuery);

... // Issue graphics commands

pDeviceContext->End(pQuery);
UINT64 queryData; // This data type is different depending on the query type

while( S_OK != pDeviceContext->GetData(pQuery, &queryData, sizeof(UINT64),
0) )
{
}

```

When using a query that does not require a call to [Begin](#), it still requires a call to [End](#). The call to [End](#) causes the data returned by [GetData](#) to be accurate up until the last call to [End](#).

## Requirements

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

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11Asynchronous](#)

---

## Feedback

Was this page helpful?

[Yes](#)

[No](#)



# ID3D11Query::GetDesc method (d3d11.h)

Article 02/22/2024

Get a query description.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_QUERY_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_QUERY\\_DESC\\*](#)

Pointer to a query description (see [D3D11\\_QUERY\\_DESC](#)).

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Query](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Query1 interface (d3d11\_3.h)

Article 02/22/2024

Represents a query object for querying information from the graphics processing unit (GPU).

## Inheritance

The [ID3D11Query1](#) interface inherits from [ID3D11Query](#). [ID3D11Query1](#) also has these types of members:

## Methods

The [ID3D11Query1](#) interface has these methods.

[+] [Expand table](#)

### [ID3D11Query1::GetDesc1](#)

Gets a query description.

## Remarks

A query can be created with [ID3D11Device3::CreateQuery1](#).

Query data is typically gathered by issuing an [ID3D11DeviceContext::Begin](#) command, issuing some graphics commands, issuing an [ID3D11DeviceContext::End](#) command, and then calling [ID3D11DeviceContext::GetData](#) to get data about what happened in between the Begin and End calls. The data returned by **GetData** will be different depending on the type of query.

There are, however, some queries that do not require calls to **Begin**. For a list of possible queries see [D3D11\\_QUERY](#).

When using a query that does not require a call to **Begin**, it still requires a call to **End**. The call to **End** causes the data returned by [GetData](#) to be accurate up until the last call to **End**.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_3.h

## See also

[Core Interfaces](#)

[ID3D11Query](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Query1::GetDesc1 method (d3d11\_3.h)

Article 02/22/2024

Gets a query description.

## Syntax

C++

```
void GetDesc1(  
    [out] D3D11_QUERY_DESC1 *pDesc1  
);
```

## Parameters

[out] pDesc1

Type: [D3D11\\_QUERY\\_DESC1\\*](#)

A pointer to a [D3D11\\_QUERY\\_DESC1](#) structure that receives a description of the query.

## Return value

None

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11Query1](#)

---

## Feedback

Was this page helpful?



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

# ID3D11RasterizerState interface (d3d11.h)

Article 02/22/2024

The rasterizer-state interface holds a description for rasterizer state that you can bind to the [rasterizer stage](#).

## Inheritance

The **ID3D11RasterizerState** interface inherits from [ID3D11DeviceChild](#).

**ID3D11RasterizerState** also has these types of members:

## Methods

The **ID3D11RasterizerState** interface has these methods.

[+] [Expand table](#)

### [ID3D11RasterizerState::GetDesc](#)

Gets the description for rasterizer state that you used to create the rasterizer-state object.  
(`ID3D11RasterizerState::GetDesc`)

## Remarks

To create a rasterizer-state object, call [ID3D11Device::CreateRasterizerState](#). To bind the rasterizer-state object to the [rasterizer stage](#), call [ID3D11DeviceContext::RSSetState](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RasterizerState::GetDesc method (d3d11.h)

Article 02/22/2024

Gets the description for rasterizer state that you used to create the rasterizer-state object.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_RASTERIZER_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_RASTERIZER\\_DESC\\*](#)

A pointer to a [D3D11\\_RASTERIZER\\_DESC](#) structure that receives a description of the rasterizer state.

## Return value

None

## Remarks

You use the description for rasterizer state in a call to the [ID3D11Device::CreateRasterizerState](#) method to create the rasterizer-state object.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11RasterizerState](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RasterizerState1 interface (d3d11\_1.h)

Article 02/22/2024

The rasterizer-state interface holds a description for rasterizer state that you can bind to the [rasterizer stage](#). This rasterizer-state interface supports forced sample count.

## Inheritance

The **ID3D11RasterizerState1** interface inherits from [ID3D11RasterizerState](#).

**ID3D11RasterizerState1** also has these types of members:

## Methods

The **ID3D11RasterizerState1** interface has these methods.

[+] [Expand table](#)

### [ID3D11RasterizerState1::GetDesc1](#)

Gets the description for rasterizer state that you used to create the rasterizer-state object.  
([ID3D11RasterizerState1::GetDesc1](#))

## Remarks

To create a rasterizer-state object, call [ID3D11Device1::CreateRasterizerState1](#). To bind the rasterizer-state object to the [rasterizer stage](#), call [ID3D11DeviceContext::RSSetState](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h

## See also

[Core Interfaces](#)

[ID3D11RasterizerState](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RasterizerState1::GetDesc1 method (d3d11\_1.h)

Article 02/22/2024

Gets the description for rasterizer state that you used to create the rasterizer-state object.

## Syntax

C++

```
void GetDesc1(  
    [out] D3D11_RASTERIZER_DESC1 *pDesc  
) ;
```

## Parameters

[out] pDesc

A pointer to a [D3D11\\_RASTERIZER\\_DESC1](#) structure that receives a description of the rasterizer state. This rasterizer state can specify forced sample count.

## Return value

None

## Remarks

You use the description for rasterizer state in a call to the [ID3D11Device1::CreateRasterizerState1](#) method to create the rasterizer-state object.

## Requirements

[ Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3D11RasterizerState1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RasterizerState2 interface (d3d11\_3.h)

Article 07/27/2022

The rasterizer-state interface holds a description for rasterizer state that you can bind to the [rasterizer stage](#). This rasterizer-state interface supports forced sample count and conservative rasterization mode.

## Inheritance

The **ID3D11RasterizerState2** interface inherits from [ID3D11RasterizerState1](#).

**ID3D11RasterizerState2** also has these types of members:

## Methods

The **ID3D11RasterizerState2** interface has these methods.

### [ID3D11RasterizerState2::GetDesc2](#)

Gets the description for rasterizer state that you used to create the rasterizer-state object.  
([ID3D11RasterizerState2::GetDesc2](#))

## Remarks

To create a rasterizer-state object, call [ID3D11Device3::CreateRasterizerState2](#). To bind the rasterizer-state object to the [rasterizer stage](#), call [ID3D11DeviceContext::RSSetState](#).

## Requirements

Minimum supported client	Windows 10 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_3.h

## See also

[Core Interfaces](#)

[ID3D11RasterizerState1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RasterizerState2::GetDesc2 method (d3d11\_3.h)

Article07/27/2022

Gets the description for rasterizer state that you used to create the rasterizer-state object.

## Syntax

C++

```
void GetDesc2(  
    [out] D3D11_RASTERIZER_DESC2 *pDesc  
);
```

## Parameters

[out] pDesc

A pointer to a [D3D11\\_RASTERIZER\\_DESC2](#) structure that receives a description of the rasterizer state. This rasterizer state can specify forced sample count and conservative rasterization mode.

## Return value

None

## Remarks

You use the description for rasterizer state in a call to the [ID3D11Device3::CreateRasterizerState2](#) method to create the rasterizer-state object.

## Requirements

Minimum supported client

Windows 10 [desktop apps only]

Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h
Library	D3D11.lib

## See also

[ID3D11RasterizerState2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11SamplerState interface (d3d11.h)

Article 02/22/2024

The sampler-state interface holds a description for sampler state that you can bind to any shader stage of the [pipeline](#) for reference by texture sample operations.

## Inheritance

The **ID3D11SamplerState** interface inherits from [ID3D11DeviceChild](#).

**ID3D11SamplerState** also has these types of members:

## Methods

The **ID3D11SamplerState** interface has these methods.

[+] [Expand table](#)

### [ID3D11SamplerState::GetDesc](#)

Gets the description for sampler state that you used to create the sampler-state object.

## Remarks

To create a sampler-state object, call [ID3D11Device::CreateSamplerState](#).

To bind a sampler-state object to any [pipeline](#) shader stage, call the following methods:

- [ID3D11DeviceContext::VSSetSamplers](#)
- [ID3D11DeviceContext::HSSetSamplers](#)
- [ID3D11DeviceContext::DSSetSamplers](#)
- [ID3D11DeviceContext::GSSetSamplers](#)
- [ID3D11DeviceContext::PSSetSamplers](#)
- [ID3D11DeviceContext::CSSetSamplers](#)

You can bind the same sampler-state object to multiple shader stages simultaneously.

## Requirements

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11SamplerState::GetDesc method (d3d11.h)

Article 02/22/2024

Gets the description for sampler state that you used to create the sampler-state object.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_SAMPLER_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_SAMPLER\\_DESC\\*](#)

A pointer to a [D3D11\\_SAMPLER\\_DESC](#) structure that receives a description of the sampler state.

## Return value

None

## Remarks

You use the description for sampler state in a call to the [ID3D11Device::CreateSamplerState](#) method to create the sampler-state object.

## Requirements

[ Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11SamplerState](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11 core functions

Article • 11/23/2019

This section contains information about the core functions.

## In this section

Topic	Description
<a href="#">D3D11CreateDevice</a>	Creates a device that represents the display adapter.
<a href="#">D3D11CreateDeviceAndSwapChain</a>	Creates a device that represents the display adapter and a swap chain used for rendering.

## Related topics

[Core Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11CreateDevice function (d3d11.h)

Article 07/27/2022

Creates a device that represents the display adapter.

## Syntax

C++

```
HRESULT D3D11CreateDevice(
    [in, optional] IDXGIAdapter          *pAdapter,
    D3D_DRIVER_TYPE                      DriverType,
    HMODULE                               Software,
    UINT                                  Flags,
    [in, optional] const D3D_FEATURE_LEVEL *pFeatureLevels,
    UINT                                  FeatureLevels,
    UINT                                  SDKVersion,
    [out, optional] ID3D11Device          **ppDevice,
    [out, optional] D3D_FEATURE_LEVEL      *pFeatureLevel,
    [out, optional] ID3D11DeviceContext   **ppImmediateContext
);
```

## Parameters

[in, optional] pAdapter

Type: [IDXGIAdapter\\*](#)

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

DriverType

Type: [D3D\\_DRIVER\\_TYPE](#)

The [D3D\\_DRIVER\\_TYPE](#), which represents the driver type to create.

## Software

Type: [HMODULE](#)

A handle to a DLL that implements a software rasterizer. If *DriverType* is *D3D\_DRIVER\_TYPE\_SOFTWARE*, *Software* must not be **NULL**. Get the handle by calling [LoadLibrary](#), [LoadLibraryEx](#), or [GetModuleHandle](#).

## Flags

Type: [UINT](#)

The runtime [layers](#) to enable (see [D3D11\\_CREATE\\_DEVICE\\_FLAG](#)); values can be bitwise OR'd together.

[in, optional] *pFeatureLevels*

Type: [const D3D\\_FEATURE\\_LEVEL\\*](#)

A pointer to an array of [D3D\\_FEATURE\\_LEVEL](#)s, which determine the order of feature levels to attempt to create. If *pFeatureLevels* is set to **NULL**, this function uses the following array of feature levels:

```
{  
    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,  
};
```

**Note** If the Direct3D 11.1 runtime is present on the computer and *pFeatureLevels* is set to **NULL**, this function won't create a *D3D\_FEATURE\_LEVEL\_11\_1* device. To create a *D3D\_FEATURE\_LEVEL\_11\_1* device, you must explicitly provide a *D3D\_FEATURE\_LEVEL* array that includes *D3D\_FEATURE\_LEVEL\_11\_1*. If you provide a *D3D\_FEATURE\_LEVEL* array that contains *D3D\_FEATURE\_LEVEL\_11\_1* on a computer that doesn't have the Direct3D 11.1 runtime installed, this function immediately fails with *E\_INVALIDARG*.

## FeatureLevels

Type: [UINT](#)

The number of elements in *pFeatureLevels*.

`SDKVersion`

Type: [UINT](#)

The SDK version; use [D3D11\\_SDK\\_VERSION](#).

`[out, optional] ppDevice`

Type: [ID3D11Device\\*\\*](#)

Returns the address of a pointer to an [ID3D11Device](#) object that represents the device created. If this parameter is **NULL**, no ID3D11Device will be returned.

`[out, optional] pFeatureLevel`

Type: [D3D\\_FEATURE\\_LEVEL\\*](#)

If successful, returns the first [D3D\\_FEATURE\\_LEVEL](#) from the *pFeatureLevels* array which succeeded. Supply **NULL** as an input if you don't need to determine which feature level is supported.

`[out, optional] ppImmediateContext`

Type: [ID3D11DeviceContext\\*\\*](#)

Returns the address of a pointer to an [ID3D11DeviceContext](#) object that represents the device context. If this parameter is **NULL**, no ID3D11DeviceContext will be returned.

## Return value

Type: [HRESULT](#)

This method can return one of the [Direct3D 11 Return Codes](#).

This method returns **E\_INVALIDARG** if you set the *pAdapter* parameter to a non-**NULL** value and the *DriverType* parameter to the [D3D\\_DRIVER\\_TYPE\\_HARDWARE](#) value.

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

This entry-point is supported by the Direct3D 11 runtime, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista (KB971644).

To create a Direct3D 11.1 device ([ID3D11Device1](#)), which is available on Windows 8, Windows Server 2012, and Windows 7 and Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, you first create a [ID3D11Device](#) with this function, and then call the [QueryInterface](#) method on the [ID3D11Device](#) object to obtain the [ID3D11Device1](#) interface.

To create a Direct3D 11.2 device ([ID3D11Device2](#)), which is available on Windows 8.1 and Windows Server 2012 R2, you first create a [ID3D11Device](#) with this function, and then call the [QueryInterface](#) method on the [ID3D11Device](#) object to obtain the [ID3D11Device2](#) interface.

Set *ppDevice* and *ppImmediateContext* to **NULL** to determine which feature level is supported by looking at *pFeatureLevel* without creating a device.

For an example, see [How To: Create a Device and Immediate Context](#); to create a device and a swap chain at the same time, use [D3D11CreateDeviceAndSwapChain](#).

If you set the *pAdapter* parameter to a non-**NULL** value, you must also set the *DriverType* parameter to the `D3D_DRIVER_TYPE_UNKNOWN` value. If you set the *pAdapter* parameter to a non-**NULL** value and the *DriverType* parameter to the `D3D_DRIVER_TYPE_HARDWARE` value, [D3D11CreateDevice](#) returns an [HRESULT](#) of `E_INVALIDARG`.

Differences between Direct3D 10 and Direct3D 11:

In Direct3D 10, the presence of *pAdapter* dictated which adapter to use and the *DriverType* could mismatch what the adapter was.

In Direct3D 11, if you are trying to create a hardware or a software device, set *pAdapter* != **NULL** which constrains the other inputs to be:

- *DriverType* must be `D3D_DRIVER_TYPE_UNKNOWN`
- *Software* must be **NULL**.

On the other hand, if *pAdapter* == **NULL**, the *DriverType* cannot be set to `D3D_DRIVER_TYPE_UNKNOWN`; it can be set to either:

- If *DriverType* == `D3D_DRIVER_TYPE_SOFTWARE`, *Software* cannot be **NULL**.
- If *DriverType* == `D3D_DRIVER_TYPE_HARDWARE`, the adapter used will be the default adapter, which is the first adapter that is enumerated by [IDXGIFactory1::EnumAdapters](#)

The function signature PFN\_D3D11\_CREATE\_DEVICE is provided as a typedef, so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

**Windows Phone 8:** This API is supported.

**Windows Phone 8.1:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib
DLL	D3D11.dll

## See also

[Core Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11CreateDeviceAndSwapChain function (d3d11.h)

Article 10/13/2021

Creates a device that represents the display adapter and a swap chain used for rendering.

## Syntax

C++

```
HRESULT D3D11CreateDeviceAndSwapChain(
    [in, optional] IDXGIAdapter              *pAdapter,
    D3D_DRIVER_TYPE                         DriverType,
    HMODULE                                 Software,
    UINT                                    Flags,
    [in, optional] const D3D_FEATURE_LEVEL   *pFeatureLevels,
    UINT                                    FeatureLevels,
    UINT                                    SDKVersion,
    [in, optional] const DXGI_SWAP_CHAIN_DESC *pSwapChainDesc,
    [out, optional] IDXGISwapChain          **ppSwapChain,
    [out, optional] ID3D11Device            **ppDevice,
    [out, optional] D3D_FEATURE_LEVEL       *pFeatureLevel,
    [out, optional] ID3D11DeviceContext     **ppImmediateContext
);
```

## Parameters

[in, optional] pAdapter

Type: [IDXGIAdapter\\*](#)

A pointer to the video adapter to use when creating a [device](#). Pass **NULL** to use the default adapter, which is the first adapter enumerated by [IDXGIFactory1::EnumAdapters](#).

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

DriverType

Type: [D3D\\_DRIVER\\_TYPE](#)

The [D3D\\_DRIVER\\_TYPE](#), which represents the driver type to create.

Software

Type: [HMODULE](#)

A handle to a DLL that implements a software rasterizer. If *DriverType* is [D3D\\_DRIVER\\_TYPE\\_SOFTWARE](#), *Software* must not be **NULL**. Get the handle by calling [LoadLibrary](#), [LoadLibraryEx](#), or [GetModuleHandle](#). The value should be non-**NULL** when [D3D\\_DRIVER\\_TYPE](#) is [D3D\\_DRIVER\\_TYPE\\_SOFTWARE](#) and **NULL** otherwise.

Flags

Type: [UINT](#)

The runtime [layers](#) to enable (see [D3D11\\_CREATE\\_DEVICE\\_FLAG](#)); values can be bitwise OR'd together.

[in, optional] *pFeatureLevels*

Type: const [D3D\\_FEATURE\\_LEVEL](#)\*

A pointer to an array of [D3D\\_FEATURE\\_LEVEL](#)s, which determine the order of feature levels to attempt to create. If *pFeatureLevels* is set to **NULL**, this function uses the following array of feature levels:

```
{  
    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,  
};
```

**Note** If the Direct3D 11.1 runtime is present on the computer and *pFeatureLevels* is set to **NULL**, this function won't create a [D3D\\_FEATURE\\_LEVEL\\_11\\_1](#) device. To create a [D3D\\_FEATURE\\_LEVEL\\_11\\_1](#) device, you must explicitly provide a [D3D\\_FEATURE\\_LEVEL](#) array that includes [D3D\\_FEATURE\\_LEVEL\\_11\\_1](#). If you provide a [D3D\\_FEATURE\\_LEVEL](#) array that contains [D3D\\_FEATURE\\_LEVEL\\_11\\_1](#) on a

computer that doesn't have the Direct3D 11.1 runtime installed, this function immediately fails with E\_INVALIDARG.

#### FeatureLevels

Type: **UINT**

The number of elements in *pFeatureLevels*.

#### SDKVersion

Type: **UINT**

The SDK version; use *D3D11\_SDK\_VERSION*.

#### [in, optional] pSwapChainDesc

Type: **const DXGI\_SWAP\_CHAIN\_DESC\***

A pointer to a swap chain description (see [DXGI\\_SWAP\\_CHAIN\\_DESC](#)) that contains initialization parameters for the swap chain.

#### [out, optional] ppSwapChain

Type: **IDXGISwapChain\*\***

Returns the address of a pointer to the [IDXGISwapChain](#) object that represents the swap chain used for rendering.

#### [out, optional] ppDevice

Type: **ID3D11Device\*\***

Returns the address of a pointer to an [ID3D11Device](#) object that represents the device created. If this parameter is **NULL**, no ID3D11Device will be returned'.

#### [out, optional] pFeatureLevel

Type: **D3D\_FEATURE\_LEVEL\***

Returns a pointer to a [D3D\\_FEATURE\\_LEVEL](#), which represents the first element in an array of feature levels supported by the device. Supply **NULL** as an input if you don't need to determine which feature level is supported.

#### [out, optional] ppImmediateContext

Type: [ID3D11DeviceContext\\*\\*](#)

Returns the address of a pointer to an [ID3D11DeviceContext](#) object that represents the device context. If this parameter is **NULL**, no [ID3D11DeviceContext](#) will be returned.

## Return value

Type: [HRESULT](#)

This method can return one of the [Direct3D 11 Return Codes](#).

This method returns [DXGI\\_ERROR\\_NOT\\_CURRENTLY\\_AVAILABLE](#) if you call it in a Session 0 process.

This method returns [E\\_INVALIDARG](#) if you set the *pAdapter* parameter to a non-**NULL** value and the *DriverType* parameter to the [D3D\\_DRIVER\\_TYPE\\_HARDWARE](#) value.

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

**Note** If you call this method in a Session 0 process, it returns [DXGI\\_ERROR\\_NOT\\_CURRENTLY\\_AVAILABLE](#).

This entry-point is supported by the Direct3D 11 runtime, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista (KB971644).

To create a Direct3D 11.1 device ([ID3D11Device1](#)), which is available on Windows 8, Windows Server 2012, and Windows 7 and Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, you first create a [ID3D11Device](#) with this function, and then call the [QueryInterface](#) method on the [ID3D11Device](#) object to obtain the [ID3D11Device1](#) interface.

To create a Direct3D 11.2 device ([ID3D11Device2](#)), which is available on Windows 8.1 and Windows Server 2012 R2, you first create a [ID3D11Device](#) with this function, and then call the [QueryInterface](#) method on the [ID3D11Device](#) object to obtain the [ID3D11Device2](#) interface.

Also, see the remarks section in [D3D11CreateDevice](#) for details about input parameter dependencies. To create a device without creating a swap chain, use the [D3D11CreateDevice](#) function.

If you set the *pAdapter* parameter to a non-**NULL** value, you must also set the *DriverType* parameter to the **D3D\_DRIVER\_TYPE\_UNKNOWN** value. If you set the *pAdapter* parameter to a non-**NULL** value and the *DriverType* parameter to the **D3D\_DRIVER\_TYPE\_HARDWARE** value, [D3D11CreateDeviceAndSwapChain](#) returns an [HRESULT](#) of **E\_INVALIDARG**.

The function signature **PFN\_D3D11\_CREATE\_DEVICE\_AND\_SWAP\_CHAIN** is provided as a **typedef**, so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

## Usage notes

**Note** The [D3D11CreateDeviceAndSwapChain](#) function does not exist for Windows Store apps. Instead, Windows Store apps use the [D3D11CreateDevice](#) function and then use the [IDXGIFactory2::CreateSwapChainForCoreWindow](#) method.

**Note** This function has not been updated to support recent additional features of swap chain creation. For the most up-to-date swap chain creation methods, refer to the methods of [IDXGIFactory2](#) (including [CreateSwapChainForHwnd](#), [CreateSwapChainForCoreWindow](#) and [CreateSwapChainForComposition](#)).

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib
DLL	D3D11.dll

## See also

[Core Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11 core structures

Article • 08/29/2023

This section contains information about the core structures.

## In this section

Topic	Description
<a href="#">D3D11_BLEND_DESC</a>	Describes the blend state that you use in a call to <a href="#">ID3D11Device::CreateBlendState</a> to create a blend-state object.
<a href="#">D3D11_BLEND_DESC1</a>	<b>Note:</b> This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems. Describes the blend state that you use in a call to <a href="#">ID3D11Device1::CreateBlendState1</a> to create a blend-state object.
<a href="#">D3D11_BOX</a>	Defines a 3D box.
<a href="#">D3D11_COUNTER_DESC</a>	Describes a counter.
<a href="#">D3D11_COUNTER_INFO</a>	Information about the video card's performance counter capabilities.
<a href="#">D3D11_DEPTH_STENCIL_DESC</a>	Describes depth-stencil state.
<a href="#">D3D11_DEPTH_STENCILOP_DESC</a>	Stencil operations that can be performed based on the results of stencil test.
<a href="#">D3D11_DRAW_INDEXED_INSTANCED_INDIRECT_ARGS</a>	Arguments for draw indexed instanced indirect.
<a href="#">D3D11_DRAW_INSTANCED_INDIRECT_ARGS</a>	Arguments for draw instanced indirect.
<a href="#">D3D11_FEATURE_DATA_ARCHITECTURE_INFO</a>	<b>Note:</b> This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems. Describes information about

Topic	Description
	Direct3D 11.1 adapter architecture.
<a href="#">D3D11_FEATURE_DATA_D3D9_OPTIONS</a>	<p><b>Note:</b> This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p> <p>Describes Direct3D 9 feature options in the current graphics driver.</p>
<a href="#">D3D11_FEATURE_DATA_D3D9_OPTIONS1</a>	Describes Direct3D 9 feature options in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_D3D9_SHADOW_SUPPORT</a>	<p><b>Note:</b> This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p> <p>Describes Direct3D 9 shadow support in the current graphics driver.</p>
<a href="#">D3D11_FEATURE_DATA_D3D9_SIMPLE_INSTANCING_SUPPORT</a>	Describes whether simple instancing is supported.
<a href="#">D3D11_FEATURE_DATA_D3D10_X_HARDWARE_OPTIONS</a>	Describes compute shader and raw and structured buffer support in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS</a>	<p><b>Note:</b> This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p> <p>Describes Direct3D 11.1 feature options in the current graphics driver.</p>
<a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS1</a>	Describes Direct3D 11.2 feature options in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS2</a>	Describes Direct3D 11.3 feature options in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS3</a>	Describes Direct3D 11.3 feature options in the current graphics driver.

Topic	Description
<a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS4</a>	Describes Direct3D 11.4 feature options in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS5</a>	Describes the level of support for shared resources in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_DISPLAYABLE</a>	Describes the level of displayable surfaces supported in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_DOUBLES</a>	Describes double data type support in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_FORMAT_SUPPORT</a>	Describes which resources are supported by the current graphics driver for a given format.
<a href="#">D3D11_FEATURE_DATA_FORMAT_SUPPORT2</a>	Describes which unordered resource options are supported by the current graphics driver for a given format.
<a href="#">D3D11_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT</a>	Describes feature data GPU virtual address support, including maximum address bits per resource and per process.
<a href="#">D3D11_FEATURE_DATA_MARKER_SUPPORT</a>	Describes whether a GPU profiling technique is supported.
<a href="#">D3D11_FEATURE_DATA_SHADER_CACHE</a>	Describes the level of shader caching supported in the current graphics driver.
<a href="#">D3D11_FEATURE_DATA_SHADER_MIN_PRECISION_SUPPORT</a>	<p><b>Note:</b> This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p> <p>Describes precision support options for shaders in the current graphics driver.</p>
<a href="#">D3D11_FEATURE_DATA_THREADING</a>	Describes the multi-threading features that are supported by the current graphics driver.

Topic	Description
<a href="#">D3D11_INPUT_ELEMENT_DESC</a>	A description of a single element for the input-assembler stage.
<a href="#">D3D11_QUERY_DATA_PIPELINE_STATISTICS</a>	Query information about graphics-pipeline activity in between calls to <a href="#">ID3D11DeviceContext::Begin</a> and <a href="#">ID3D11DeviceContext::End</a> .
<a href="#">D3D11_QUERY_DATA_SO_STATISTICS</a>	Query information about the amount of data streamed out to the stream-output buffers in between <a href="#">ID3D11DeviceContext::Begin</a> and <a href="#">ID3D11DeviceContext::End</a> .
<a href="#">D3D11_QUERY_DATA_TIMESTAMP_DISJOINT</a>	Query information about the reliability of a timestamp query.
<a href="#">D3D11_QUERY_DESC</a>	Describes a query.
<a href="#">D3D11_QUERY_DESC1</a>	Describes a query.
<a href="#">D3D11_RASTERIZER_DESC</a>	Describes rasterizer state.
<a href="#">D3D11_RASTERIZER_DESC1</a>	<p><b>Note:</b> This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p> <p>Describes rasterizer state.</p>
<a href="#">D3D11_RASTERIZER_DESC2</a>	Describes rasterizer state.
<a href="#">D3D11_RECT</a>	D3D11_RECT is declared as follows:
<a href="#">D3D11_RENDER_TARGET_BLEND_DESC</a>	Describes the blend state for a render target.
<a href="#">D3D11_RENDER_TARGET_BLEND_DESC1</a>	<p><b>Note:</b> This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p> <p>Describes the blend state for a render target.</p>
<a href="#">D3D11_SAMPLER_DESC</a>	Describes a sampler state.
<a href="#">D3D11_SO_DECLARATION_ENTRY</a>	Description of a vertex element in

Topic	Description
	a vertex buffer in an output slot.
D3D11_VIEWPORT	Defines the dimensions of a viewport.

In addition, there's a 2D rectangle structure defined in D3D11.h.

C++

```
typedef RECT D3D11_RECT;
```

For documentation, see [RECT](#) in [Windows GDI](#).

## Related topics

[Core Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BLEND\_DESC structure (d3d11.h)

Article 02/22/2024

Describes the blend state that you use in a call to [ID3D11Device::CreateBlendState](#) to create a blend-state object.

## Syntax

C++

```
typedef struct D3D11_BLEND_DESC {
    BOOL           AlphaToCoverageEnable;
    BOOL           IndependentBlendEnable;
    D3D11_RENDER_TARGET_BLEND_DESC RenderTarget[8];
} D3D11_BLEND_DESC;
```

## Members

`AlphaToCoverageEnable`

Type: [BOOL](#)

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`

Type: [BOOL](#)

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.

`RenderTarget[8]`

Type: [D3D11\\_RENDER\\_TARGET\\_BLEND\\_DESC\[8\]](#)

An array of [D3D11\\_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

Here are the default values for blend state.

[+] Expand table

State	Default Value
AlphaToCoverageEnable	FALSE
IndependentBlendEnable	FALSE
RenderTarget[0].BlendEnable	FALSE
RenderTarget[0].SrcBlend	D3D11_BLEND_ONE
RenderTarget[0].DestBlend	D3D11_BLEND_ZERO
RenderTarget[0].BlendOp	D3D11_BLEND_OP_ADD
RenderTarget[0].SrcBlendAlpha	D3D11_BLEND_ONE
RenderTarget[0].DestBlendAlpha	D3D11_BLEND_ZERO
RenderTarget[0].BlendOpAlpha	D3D11_BLEND_OP_ADD
RenderTarget[0].RenderTargetWriteMask	D3D11_COLOR_WRITE_ENABLE_ALL

**Note** `D3D11_BLEND_DESC` is identical to [D3D10\\_BLEND\\_DESC1](#).

If the driver type is set to [D3D\\_DRIVER\\_TYPE\\_HARDWARE](#), the feature level is set to less than or equal to [D3D\\_FEATURE\\_LEVEL\\_9\\_3](#), and the pixel format of the render target is set to [DXGI\\_FORMAT\\_R8G8B8A8\\_UNORM\\_SRGB](#), [DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM\\_SRGB](#), or [DXGI\\_FORMAT\\_B8G8R8X8\\_UNORM\\_SRGB](#), the display device performs the blend in standard RGB (sRGB) space and not in linear space. However, if the feature level is set to greater than [D3D\\_FEATURE\\_LEVEL\\_9\\_3](#), the display device performs the blend in linear space, which is ideal.

# Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# CD3D11\_BLEND\_DESC1 structure (d3d11\_1.h)

Article 02/22/2024

Describes the blend state that you use in a call to [D3D11Device1::CreateBlendState1](#) to create a blend-state object.

## ⓘ Note

This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

## Syntax

C++

```
struct CD3D11_BLEND_DESC1 : D3D11_BLEND_DESC1 {
    void CD3D11_BLEND_DESC1();
    void CD3D11_BLEND_DESC1(
        const D3D11_BLEND_DESC1 & o
    );
    void CD3D11_BLEND_DESC1(
        CD3D11_DEFAULT unnamedParam1
    );
    void ~CD3D11_BLEND_DESC1();
};
```

## Inheritance

The `CD3D11_BLEND_DESC1` structure implements `D3D11_BLEND_DESC1`.

## Members

```
void CD3D11_BLEND_DESC1()
```

```
void CD3D11_BLEND_DESC1( const D3D11_BLEND_DESC1 & o)
```

```
void CD3D11_BLEND_DESC1( CD3D11_DEFAULT unnamedParam1)
```

```
void ~CD3D11_BLEND_DESC1()
```

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11_1.h

## See also

[Core Structures](#)

[D3D11\\_RENDER\\_TARGET\\_BLEND\\_DESC1](#)

[ID3D11BlendState1::GetDesc1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BOX structure (d3d11.h)

Article 07/27/2022

Defines a 3D box.

## Syntax

C++

```
typedef struct D3D11_BOX {
    UINT left;
    UINT top;
    UINT front;
    UINT right;
    UINT bottom;
    UINT back;
} D3D11_BOX;
```

## Members

`left`

Type: **UINT**

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

`top`

Type: **UINT**

The y position of the top of the box.

`front`

Type: **UINT**

The z position of the front of the box.

`right`

Type: **UINT**

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

`bottom`

Type: [UINT](#)

The y position of the bottom of the box.

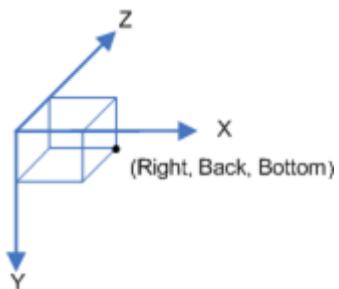
`back`

Type: [UINT](#)

The z position of the back of the box.

## Remarks

The following diagram shows a 3D box, where the origin is the left, front, top corner.



The values for **right**, **bottom**, and **back** are each one pixel past the end of the pixels that are included in the box region. That is, the values for **left**, **top**, and **front** are included in the box region while the values for right, bottom, and back are excluded from the box region. For example, for a box that is one pixel wide,  $(\text{right} - \text{left}) == 1$ ; the box region includes the left pixel but not the right pixel.

Coordinates of a box are in bytes for buffers and in texels for textures.

## Requirements

Header

d3d11.h

## See also

[Core Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3D11\_COUNTER\_DESC structure (d3d11.h)

Article 02/22/2024

Describes a counter.

## Syntax

C++

```
typedef struct D3D11_COUNTER_DESC {
    D3D11_COUNTER Counter;
    UINT          MiscFlags;
} D3D11_COUNTER_DESC;
```

## Members

Counter

Type: [D3D11\\_COUNTER](#)

Type of counter (see [D3D11\\_COUNTER](#)).

MiscFlags

Type: [UINT](#)

Reserved.

## Remarks

This structure is used by [ID3D11Counter::GetDesc](#), [ID3D11Device::CheckCounter](#) and [ID3D11Device::CreateCounter](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_COUNTER\_INFO structure (d3d11.h)

Article 02/22/2024

Information about the video card's performance counter capabilities.

## Syntax

C++

```
typedef struct D3D11_COUNTER_INFO {
    D3D11_COUNTER LastDeviceDependentCounter;
    UINT          NumSimultaneousCounters;
    UINT8         NumDetectableParallelUnits;
} D3D11_COUNTER_INFO;
```

## Members

LastDeviceDependentCounter

Type: [D3D11\\_COUNTER](#)

Largest device-dependent counter ID that the device supports. If none are supported, this value will be 0. Otherwise it will be greater than or equal to D3D11\_COUNTER\_DEVICE\_DEPENDENT\_0. See [D3D11\\_COUNTER](#).

NumSimultaneousCounters

Type: [UINT](#)

Number of counters that can be simultaneously supported.

NumDetectableParallelUnits

Type: [UINT8](#)

Number of detectable parallel units that the counter is able to discern. Values are 1 ~ 4. Use NumDetectableParallelUnits to interpret the values of the VERTEX\_PROCESSING, GEOMETRY\_PROCESSING, PIXEL\_PROCESSING, and OTHER\_GPU\_PROCESSING counters.

## Remarks

This structure is returned by [ID3D11Device::CheckCounterInfo](#).

# Requirements

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

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_DEPTH\_STENCIL\_DESC structure (d3d11.h)

Article 07/27/2022

Describes depth-stencil state.

## Syntax

C++

```
typedef struct D3D11_DEPTH_STENCIL_DESC {
    BOOL DepthEnable;
    D3D11_DEPTH_WRITE_MASK DepthWriteMask;
    D3D11_COMPARISON_FUNC DepthFunc;
    BOOL StencilEnable;
    UINT8 StencilReadMask;
    UINT8 StencilWriteMask;
    D3D11_DEPTH_STENCILOP_DESC FrontFace;
    D3D11_DEPTH_STENCILOP_DESC BackFace;
} D3D11_DEPTH_STENCIL_DESC;
```

## Members

DepthEnable

Type: [BOOL](#)

Enable depth testing.

DepthWriteMask

Type: [D3D11\\_DEPTH\\_WRITE\\_MASK](#)

Identify a portion of the depth-stencil buffer that can be modified by depth data (see [D3D11\\_DEPTH\\_WRITE\\_MASK](#)).

DepthFunc

Type: [D3D11\\_COMPARISON\\_FUNC](#)

A function that compares depth data against existing depth data. The function options are listed in [D3D11\\_COMPARISON\\_FUNC](#).

`StencilEnable`

Type: **BOOL**

Enable stencil testing.

`StencilReadMask`

Type: **UINT8**

Identify a portion of the depth-stencil buffer for reading stencil data.

`StencilWriteMask`

Type: **UINT8**

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

`FrontFace`

Type: **D3D11\_DEPTH\_STENCIL\_DESC**

Identify how to use the results of the depth test and the stencil test for pixels whose surface normal is facing towards the camera (see [D3D11\\_DEPTH\\_STENCIL\\_DESC](#)).

`BackFace`

Type: **D3D11\_DEPTH\_STENCIL\_DESC**

Identify how to use the results of the depth test and the stencil test for pixels whose surface normal is facing away from the camera (see [D3D11\\_DEPTH\\_STENCIL\\_DESC](#)).

## Remarks

Pass a pointer to **D3D11\_DEPTH\_STENCIL\_DESC** to the [ID3D11Device::CreateDepthStencilState](#) method to create the depth-stencil state object.

Depth-stencil state controls how depth-stencil testing is performed by the output-merger stage.

The following table shows the default values of depth-stencil states.

State	Default Value
DepthEnable	TRUE
DepthWriteMask	D3D11_DEPTH_WRITE_MASK_ALL

DepthFunc	D3D11_COMPARISON_LESS
StencilEnable	FALSE
StencilReadMask	D3D11_DEFAULT_STENCIL_READ_MASK
StencilWriteMask	D3D11_DEFAULT_STENCIL_WRITE_MASK
FrontFace.StencilFunc and  BackFace.StencilFunc	D3D11_COMPARISON_ALWAYS
FrontFace.StencilDepthFailOp and  BackFace.StencilDepthFailOp	D3D11_STENCIL_OP_KEEP
FrontFace.StencilPassOp and  BackFace.StencilPassOp	D3D11_STENCIL_OP_KEEP
FrontFace.StencilFailOp and  BackFace.StencilFailOp	D3D11_STENCIL_OP_KEEP

The formats that support stenciling are DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT and DXGI\_FORMAT\_D32\_FLOAT\_S8X24\_UINT.

## Requirements

Header	d3d11.h
--------	---------

## See also

[Core Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_DEPTH\_STENCILOP\_DESC structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_DEPTH_STENCILOP_DESC {
    D3D11_STENCIL_OP      StencilFailOp;
    D3D11_STENCIL_OP      StencilDepthFailOp;
    D3D11_STENCIL_OP      StencilPassOp;
    D3D11_COMPARISON_FUNC StencilFunc;
} D3D11_DEPTH_STENCILOP_DESC;
```

## Members

`StencilFailOp`

Type: [D3D11\\_STENCIL\\_OP](#)

The stencil operation to perform when stencil testing fails.

`StencilDepthFailOp`

Type: [D3D11\\_STENCIL\\_OP](#)

The stencil operation to perform when stencil testing passes and depth testing fails.

`StencilPassOp`

Type: [D3D11\\_STENCIL\\_OP](#)

The stencil operation to perform when stencil testing and depth testing both pass.

`StencilFunc`

Type: [D3D11\\_COMPARISON\\_FUNC](#)

A function that compares stencil data against existing stencil data. The function options are listed in [D3D11\\_COMPARISON\\_FUNC](#).

## Remarks

All stencil operations are specified as a [D3D11\\_STENCIL\\_OP](#). The 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).

This structure is a member of a [depth-stencil description](#).

## Requirements

  [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_DRAW\_INDEXED\_INSTANCED\_IN DIRECT\_ARGS structure (d3d11.h)

Article04/02/2021

Arguments for draw indexed instanced indirect.

## Syntax

C++

```
typedef struct D3D11_DRAW_INDEXED_INSTANCED_INDIRECT_ARGS {
    UINT IndexCountPerInstance;
    UINT InstanceCount;
    UINT StartIndexLocation;
    INT  BaseVertexLocation;
    UINT StartInstanceLocation;
} D3D11_DRAW_INDEXED_INSTANCED_INDIRECT_ARGS;
```

## 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 [ID3D11DeviceContext::DrawIndexedInstanced](#).

## Requirements

Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?



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

# D3D11\_DRAW\_INSTANCED\_INDIRECT\_ARGS structure (d3d11.h)

Article 02/22/2024

Arguments for draw instanced indirect.

## Syntax

C++

```
typedef struct D3D11_DRAW_INSTANCED_INDIRECT_ARGS {
    UINT VertexCountPerInstance;
    UINT InstanceCount;
    UINT StartVertexLocation;
    UINT StartInstanceLocation;
} D3D11_DRAW_INSTANCED_INDIRECT_ARGS;
```

## Members

`VertexCountPerInstance`

The number of vertices to draw.

`InstanceCount`

The number of instances to draw.

`StartVertexLocation`

The index of the first vertex.

`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 [ID3D11DeviceContext::DrawInstanced](#).

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_ARCHITECTURE\_INF0 structure (d3d11.h)

Article02/22/2024

**Note** This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

Describes information about Direct3D 11.1 adapter architecture.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_ARCHITECTURE_INFO {
    BOOL TileBasedDeferredRenderer;
} D3D11_FEATURE_DATA_ARCHITECTURE_INFO;
```

## Members

`TileBasedDeferredRenderer`

Specifies whether a rendering device batches rendering commands and performs multipass rendering into tiles or bins over a render area. Certain API usage patterns that are fine for TileBasedDefferredRenderers (TBDRs) can perform worse on non-TBDRs and vice versa. Applications that are careful about rendering can be friendly to both TBDR and non-TBDR architectures. **TRUE** if the rendering device batches rendering commands and **FALSE** otherwise.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

[Core Structures](#)

[D3D11\\_FEATURE](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_D3D9\_OPTIONS structure (d3d11.h)

Article07/27/2022

**Note** This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

Describes Direct3D 9 feature options in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D9_OPTIONS {
    BOOL FullNonPow2TextureSupport;
} D3D11_FEATURE_DATA_D3D9_OPTIONS;
```

## Members

`FullNonPow2TextureSupport`

Specifies whether the driver supports the nonpowers-of-2-unconditionally feature. For more information about this feature, see [feature level](#). The runtime sets this member to **TRUE** for hardware at Direct3D 10 and higher feature levels. For hardware at Direct3D 9.3 and lower feature levels, the runtime sets this member to **FALSE** if the hardware and driver support the powers-of-2 (2D textures must have widths and heights specified as powers of two) feature or the nonpowers-of-2-conditionally feature. For more information about this feature, see [feature level](#).

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]

Header

d3d11.h

## See also

[Core Structures](#)

[D3D11\\_FEATURE](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_D3D9\_OPTIONS1 structure (d3d11.h)

Article07/27/2022

**Note** This structure is supported by the Direct3D 11.2 runtime, which is available on Windows 8.1 and later operating systems.

Describes Direct3D 9 feature options in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D9_OPTIONS1 {
    BOOL FullNonPow2TextureSupported;
    BOOL DepthAsTextureWithLessEqualComparisonFilterSupported;
    BOOL SimpleInstancingSupported;
    BOOL TextureCubeFaceRenderTargetWithNonCubeDepthStencilSupported;
} D3D11_FEATURE_DATA_D3D9_OPTIONS1;
```

## Members

`FullNonPow2TextureSupported`

Specifies whether the driver supports the nonpowers-of-2-unconditionally feature. For more info about this feature, see [feature level](#). The runtime sets this member to **TRUE** for hardware at Direct3D 10 and higher feature levels. For hardware at Direct3D 9.3 and lower feature levels, the runtime sets this member to **FALSE** if the hardware and driver support the powers-of-2 (2D textures must have widths and heights specified as powers of two) feature or the nonpowers-of-2-conditionally feature.

`DepthAsTextureWithLessEqualComparisonFilterSupported`

Specifies whether the driver supports the shadowing feature with the comparison-filtering mode set to less than or equal to. The runtime sets this member to **TRUE** for hardware at Direct3D 10 and higher [feature levels](#). For hardware at Direct3D 9.3 and lower feature levels, the runtime sets this member to **TRUE** only if the hardware and driver support the shadowing feature; otherwise **FALSE**.

### **SimpleInstancingSupported**

Specifies whether the hardware and driver support simple instancing. The runtime sets this member to **TRUE** if the hardware and driver support simple instancing.

### **TextureCubeFaceRenderTargetWithNonCubeDepthStencilSupported**

Specifies whether the hardware and driver support setting a single face of a [TextureCube](#) as a render target while the depth stencil surface that is bound alongside can be a [Texture2D](#) (as opposed to [TextureCube](#)). The runtime sets this member to **TRUE** if the hardware and driver support this feature; otherwise **FALSE**.

If the hardware and driver don't support this feature, the app must match the render target surface type with the depth stencil surface type. Because hardware at Direct3D 9.3 and lower [feature levels](#) doesn't allow [TextureCube](#) depth surfaces, the only way to render a scene into a [TextureCube](#) while having depth buffering enabled is to render each [TextureCube](#) face separately to a [Texture2D](#) render target first (because that can be matched with a [Texture2D](#) depth), and then copy the results into the [TextureCube](#). If the hardware and driver support this feature, the app can just render to the [TextureCube](#) faces directly while getting depth buffering out of a [Texture2D](#) depth buffer.

You only need to query this feature from hardware at Direct3D 9.3 and lower [feature levels](#) because hardware at Direct3D 10.0 and higher feature levels allow [TextureCube](#) depth surfaces.

## Remarks

You can use the [D3D11\\_FEATURE\\_D3D9\\_OPTIONS1](#) enumeration value with [ID3D11Device::CheckFeatureSupport](#) to query a driver about support for Direct3D 9 feature options rather than making multiple calls to [ID3D11Device::CheckFeatureSupport](#) by using [D3D11\\_FEATURE\\_D3D9\\_OPTIONS](#), [D3D11\\_FEATURE\\_D3D9\\_SHADOW\\_SUPPORT](#), and [D3D11\\_FEATURE\\_D3D9\\_SIMPLE\\_INSTANCING\\_SUPPORT](#), which provide identical info about supported Direct3D 9 feature options.

## Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]

## See also

[Core Structures](#)

[D3D11\\_FEATURE](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_D3D9\_SHADOW\_SUPPORT structure (d3d11.h)

Article04/02/2021

**Note** This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

Describes Direct3D 9 shadow support in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D9_SHADOW_SUPPORT {
    BOOL SupportsDepthAsTextureWithLessEqualComparisonFilter;
} D3D11_FEATURE_DATA_D3D9_SHADOW_SUPPORT;
```

## Members

`SupportsDepthAsTextureWithLessEqualComparisonFilter`

Specifies whether the driver supports the shadowing feature with the comparison-filtering mode set to less than or equal to. The runtime sets this member to **TRUE** for hardware at Direct3D 10 and higher [feature levels](#). For hardware at Direct3D 9.3 and lower feature levels, the runtime sets this member to **TRUE** only if the hardware and driver support the shadowing feature; otherwise **FALSE**.

## Remarks

Shadows are an important element in realistic 3D scenes. You can use the shadow buffer technique to render shadows. The basic principle of the technique is to use a depth buffer to store the scene depth info from the perspective of the light source, and then compare each point rendered in the scene with that buffer to determine if it is in shadow.

To render objects into the scene with shadows on them, you create [sampler state objects](#) with comparison filtering set and the comparison mode (ComparisonFunc) to

LessEqual. You can also set BorderColor addressing on this depth sampler, even though BorderColor isn't typically allowed on [feature levels](#) 9.1 and 9.2. By using the border color and picking 0.0 or 1.0 as the border color value, you can control whether the regions off the edge of the shadow map appear to be always in shadow or never in shadow respectively. You can control the shadow filter quality by the Mag and Min filter settings in the comparison sampler. Point sampling will produce shadows with non-anti-aliased edges. Linear filter sampler settings will result in higher quality shadow edges, but might affect performance on some power-optimized devices.

**Note** If you use a separate setting for Mag versus Min filter options, you produce an undefined result. Anisotropic filtering is not supported. The Mip filter choice is not relevant because **feature level** 9.x does not allow mipmapped depth buffers.

**Note** On **feature level** 9.x, you can't compile a shader with the **SampleCmp** and **SampleCmpLevelZero** intrinsic functions by using older versions of the compiler. For example, you can't use the **fxc.exe** compiler that ships with the DirectX SDK or use the **D3DCompile\*\*** functions (like **D3DCompileFromFile**) that are implemented in D3DCompiler\_43.dll and earlier. These intrinsic functions on feature level 9.x are only supported in the fxc.exe compiler that ships with the Windows 8 SDK and later and with the **D3DCompile\*\*** functions that are implemented in D3DCompiler\_44.dll and later. But these intrinsic functions are present in shader models for feature levels higher than 9.x.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_D3D9\_SIMPLE\_INSTANCING\_SUPPORT structure (d3d11.h)

Article04/02/2021

**Note** This structure is supported by the Direct3D 11.2 runtime, which is available on Windows 8.1 and later operating systems.

Describes whether simple instancing is supported.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D9_SIMPLE_INSTANCING_SUPPORT {
    BOOL SimpleInstancingSupported;
} D3D11_FEATURE_DATA_D3D9_SIMPLE_INSTANCING_SUPPORT;
```

## Members

`SimpleInstancingSupported`

Specifies whether the hardware and driver support simple instancing. The runtime sets this member to **TRUE** if the hardware and driver support simple instancing.

## Remarks

If the Direct3D API is the Direct3D 11.2 runtime and can support 11.2 features, [ID3D11Device::CheckFeatureSupport](#) for `D3D11_FEATURE_D3D9_SIMPLE_INSTANCING_SUPPORT` will return a **SUCCESS** code when valid parameters are passed. The `SimpleInstancingSupported` member of `D3D11_FEATURE_DATA_D3D9_SIMPLE_INSTANCING_SUPPORT` will be set to **TRUE** or **FALSE**.

Simple instancing means that instancing is supported with the caveat that the `InstanceDataStepRate` member of the [D3D11\\_INPUT\\_ELEMENT\\_DESC](#) structure must be equal to 1. This does not change the full instancing support provided by hardware at

feature level 9.3 and above, and is meant to expose the instancing support that may be available on feature level 9.2 and 9.1 hardware.

## Requirements

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

[Core Structures](#)

[D3D11\\_FEATURE](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_D3D10\_X\_HARDWARE\_OPTIONS structure (d3d11.h)

Article 02/22/2024

Describes compute shader and raw and structured buffer support in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D10_X_HARDWARE_OPTIONS {
    BOOL ComputeShaders_Plus_RawAndStructuredBuffers_Via_Shader_4_x;
} D3D11_FEATURE_DATA_D3D10_X_HARDWARE_OPTIONS;
```

## Members

ComputeShaders\_Plus\_RawAndStructuredBuffers\_Via\_Shader\_4\_x

Type: **BOOL**

TRUE if compute shaders and raw and structured buffers are supported; otherwise FALSE.

## Remarks

Direct3D 11 devices (D3D\_FEATURE\_LEVEL\_11\_0) are required to support Compute Shader model 5.0. Direct3D 10.x devices (D3D\_FEATURE\_LEVEL\_10\_0, D3D\_FEATURE\_LEVEL\_10\_1) can optionally support Compute Shader model 4.0 or 4.1.

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS structure (d3d11.h)

Article04/02/2021

Describes Direct3D 11.1 feature options in the current graphics driver.

## ⓘ Note

This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D11_OPTIONS {
    BOOL OutputMergerLogicOp;
    BOOL UAVOnlyRenderingForcedSampleCount;
    BOOL DiscardAPIsSeenByDriver;
    BOOL FlagsForUpdateAndCopySeenByDriver;
    BOOL ClearView;
    BOOL CopyWithOverlap;
    BOOL ConstantBufferPartialUpdate;
    BOOL ConstantBufferOffsetting;
    BOOL MapNoOverwriteOnDynamicConstantBuffer;
    BOOL MapNoOverwriteOnDynamicBufferSRV;
    BOOL MultisampleRTVWithForcedSampleCountOne;
    BOOL SAD4ShaderInstructions;
    BOOL ExtendedDoublesShaderInstructions;
    BOOL ExtendedResourceSharing;
} D3D11_FEATURE_DATA_D3D11_OPTIONS;
```

## Members

`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.

`UAVOnlyRenderingForcedSampleCount`

Specifies whether the driver can render with no render target views (RTVs) or depth stencil views (DSVs), and only unordered access views (UAVs) bound. The runtime sets this member to **TRUE** if the driver can render with no RTVs or DSVs and only UAVs bound and **FALSE** otherwise. If **TRUE**, you can set the **ForcedSampleCount** member of **D3D11\_RASTERIZER\_DESC1** to 1, 4, or 8 when you render with no RTVs or DSV and only UAVs bound. For **feature level** 11.1, this member is always **TRUE** and you can also set **ForcedSampleCount** to 16 in addition to 1, 4, or 8. The default value of **ForcedSampleCount** is 0, which means the same as if the value is set to 1. You can always set **ForcedSampleCount** to 0 or 1 for UAV-only rendering independently of how this member is set.

#### `DiscardAPIsSeenByDriver`

Specifies whether the driver supports the [ID3D11DeviceContext1::DiscardView](#) and [ID3D11DeviceContext1::DiscardResource](#) methods. The runtime sets this member to **TRUE** if the driver supports these methods and **FALSE** otherwise. How this member is set does not indicate whether the driver actually uses these methods; that is, the driver might ignore these methods if they are not useful to the hardware. If **FALSE**, the runtime does not expose these methods to the driver because the driver does not support them. You can monitor this member during development to rule out legacy drivers on hardware where these methods might have otherwise been beneficial. You are not required to write separate code paths based on whether this member is **TRUE** or **FALSE**; you can call these methods whenever applicable.

#### `FlagsForUpdateAndCopySeenByDriver`

Specifies whether the driver supports new semantics for copy and update that are exposed by the [ID3D11DeviceContext1::CopySubresourceRegion1](#) and [ID3D11DeviceContext1::UpdateSubresource1](#) methods. The runtime sets this member to **TRUE** if the driver supports new semantics for copy and update. The runtime sets this member to **FALSE** only for legacy drivers. The runtime handles this member similarly to the **DiscardAPIsSeenByDriver** member.

#### `ClearView`

Specifies whether the driver supports the [ID3D11DeviceContext1::ClearView](#) method. The runtime sets this member to **TRUE** if the driver supports this method and **FALSE** otherwise. If **FALSE**, the runtime does not expose this method to the driver because the driver does not support it.

**Note** For **feature level** 9.1, 9.2, and 9.3, this member is always **TRUE** because the option is emulated by the runtime.

#### CopyWithOverlap

Specifies whether you can call [ID3D11DeviceContext1::CopySubresourceRegion1](#) with overlapping source and destination rectangles. The runtime sets this member to **TRUE** if you can call **CopySubresourceRegion1** with overlapping source and destination rectangles and **FALSE** otherwise. If **FALSE**, the runtime does not expose this method to the driver because the driver does not support it.

**Note** For feature level 9.1, 9.2, and 9.3, this member is always **TRUE** because drivers already support the option for these feature levels.

#### ConstantBufferPartialUpdate

Specifies whether the driver supports partial updates of constant buffers. The runtime sets this member to **TRUE** if the driver supports partial updates of constant buffers and **FALSE** otherwise. If **FALSE**, the runtime does not expose this operation to the driver because the driver does not support it.

**Note** For feature level 9.1, 9.2, and 9.3, this member is always **TRUE** because the option is emulated by the runtime.

#### ConstantBufferOffsetting

Specifies whether the driver supports new semantics for setting offsets in constant buffers for a shader. The runtime sets this member to **TRUE** if the driver supports allowing you to specify offsets when you call new methods like the [ID3D11DeviceContext1::VSSetConstantBuffers1](#) method and **FALSE** otherwise. If **FALSE**, the runtime does not expose this operation to the driver because the driver does not support it.

**Note** For feature level 9.1, 9.2, and 9.3, this member is always **TRUE** because the option is emulated by the runtime.

#### MapNoOverwriteOnDynamicConstantBuffer

Specifies whether you can call [ID3D11DeviceContext::Map](#) with [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#) on a dynamic constant buffer (that is, whether the driver supports this operation). The runtime sets this member to **TRUE** if the driver supports this operation and **FALSE** otherwise. If **FALSE**, the runtime fails this method because the driver does not support the operation.

**Note** For feature level 9.1, 9.2, and 9.3, this member is always **TRUE** because the option is emulated by the runtime.

#### MapNoOverwriteOnDynamicBufferSRV

Specifies whether you can call [ID3D11DeviceContext::Map](#) with [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#) on a dynamic buffer SRV (that is, whether the driver supports this operation). The runtime sets this member to **TRUE** if the driver supports this operation and **FALSE** otherwise. If **FALSE**, the runtime fails this method because the driver does not support the operation.

#### MultisampleRTVWithForcedSampleCountOne

Specifies whether the driver supports multisample rendering when you render with RTVs bound. If **TRUE**, you can set the **ForcedSampleCount** member of [D3D11\\_RASTERIZER\\_DESC1](#) to 1 with a multisample RTV bound. The driver can support this option on [feature level](#) 10 and higher. If **FALSE**, the rasterizer-state creation will fail because the driver is legacy or the feature level is too low.

#### SAD4ShaderInstructions

Specifies whether the hardware and driver support the [msad4](#) intrinsic function in shaders. The runtime sets this member to **TRUE** if the hardware and driver support calls to [msad4](#) intrinsic functions in shaders. If **FALSE**, the driver is legacy or the hardware does not support the option; the runtime will fail shader creation for shaders that use [msad4](#).

#### ExtendedDoublesShaderInstructions

Specifies whether the hardware and driver support the [fma](#) intrinsic function and other extended doubles instructions ([DDIV](#) and [DRCP](#)) in shaders. The [fma](#) intrinsic function emits an extended doubles [DFMA](#) instruction. The runtime sets this member to **TRUE** if the hardware and driver support extended doubles instructions in shaders ([shader model 5](#) and higher). Support of this option implies support of basic double-precision shader instructions as well. You can use the [D3D11\\_FEATURE\\_DOUBLES](#) value to query

for support of double-precision shaders. If **FALSE**, the hardware and driver do not support the option; the runtime will fail shader creation for shaders that use extended doubles instructions.

#### ExtendedResourceSharing

Specifies whether the hardware and driver have [extended support for shared Texture2D resource types and formats](#). The runtime sets this member to **TRUE** if the hardware and driver support extended Texture2D resource sharing.

## Remarks

If a Microsoft Direct3D device supports [feature level 11.1 \(D3D\\_FEATURE\\_LEVEL\\_11\\_1\)](#), when you call [ID3D11Device::CheckFeatureSupport](#) with [D3D11\\_FEATURE\\_D3D11\\_OPTIONS](#), [CheckFeatureSupport](#) returns a pointer to [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) with all member set to **TRUE** except the **SAD4ShaderInstructions** and **ExtendedDoublesShaderInstructions** members, which are optionally supported by the hardware and driver and therefore can be **TRUE** or **FALSE**.

[Feature level](#) 11.1 provides the following additional features:

- UAVs at every shader stage with 64 UAV bind slots instead of 8.
- Target-independent rasterization, which enables you to set the **ForcedSampleCount** member of [D3D11\\_RASTERIZER\\_DESC1](#) to 1, 4, 8, or 16 and to render to RTVs with a single sample.
- UAV-only rendering with the **ForcedSampleCount** member of [D3D11\\_RASTERIZER\\_DESC1](#) set to up to 16 (only up to 8 for [feature level](#) 11).

The runtime always sets the following groupings of members identically. That is, all the values in a grouping are **TRUE** or **FALSE** together:

- **DiscardAPIsSeenByDriver** and **FlagsForUpdateAndCopySeenByDriver**
- **ClearView**, **CopyWithOverlap**, **ConstantBufferPartialUpdate**, **ConstantBufferOffsetting**, and **MapNoOverwriteOnDynamicConstantBuffer**
- **MapNoOverwriteOnDynamicBufferSRV** and **MultisampleRTVWithForcedSampleCountOne**

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

[Core Structures](#)

[D3D11\\_FEATURE](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS1 structure (d3d11.h)

Article04/02/2021

## ⓘ Note

This structure is supported by the Direct3D 11.2 runtime, which is available on Windows 8.1 and later operating systems.

Describes Direct3D 11.2 feature options in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D11_OPTIONS1 {
    D3D11_TILED_RESOURCES_TIER TiledResourcesTier;
    BOOL MinMaxFiltering;
    BOOL ClearViewAlsoSupportsDepthOnlyFormats;
    BOOL MapOnDefaultBuffers;
} D3D11_FEATURE_DATA_D3D11_OPTIONS1;
```

## Members

TiledResourcesTier

Type: [D3D11\\_TILED\\_RESOURCES\\_TIER](#)

Specifies whether the hardware and driver support tiled resources. The runtime sets this member to a [D3D11\\_TILED\\_RESOURCES\\_TIER](#)-typed value that indicates if the hardware and driver support tiled resources and at what tier level.

MinMaxFiltering

Type: [BOOL](#)

Specifies whether the hardware and driver support the filtering options ([D3D11\\_FILTER](#)) of comparing the result to the minimum or maximum value during texture sampling. The runtime sets this member to **TRUE** if the hardware and driver support these filtering options.

`ClearViewAlsoSupportsDepthOnlyFormats`

Type: **BOOL**

Specifies whether the hardware and driver also support the [ID3D11DeviceContext1::ClearView](#) method on depth formats. For info about valid depth formats, see [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#).

`MapOnDefaultBuffers`

Type: **BOOL**

Specifies support for creating [ID3D11Buffer](#) resources that can be passed to the [ID3D11DeviceContext::Map](#) and [ID3D11DeviceContext::Unmap](#) methods. This means that the **CPUAccessFlags** member of the [D3D11\\_BUFFER\\_DESC](#) structure may be set with the desired [D3D11\\_CPU\\_ACCESS\\_FLAG](#) elements when the **Usage** member of [D3D11\\_BUFFER\\_DESC](#) is set to [D3D11\\_USAGE\\_DEFAULT](#). The runtime sets this member to **TRUE** if the hardware is capable of at least [D3D\\_FEATURE\\_LEVEL\\_11\\_0](#) and the graphics device driver supports mappable default buffers.

## Remarks

If the Direct3D API is the Direct3D 11.2 runtime and can support 11.2 features, [ID3D11Device::CheckFeatureSupport](#) for [D3D11\\_FEATURE\\_D3D11\\_OPTIONS1](#) will return a **SUCCESS** code when valid parameters are passed. The members of [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS1](#) will be set appropriately based on the system's graphics hardware and graphics driver.

## Mappable default buffers

When creating a default buffer with [D3D11\\_CPU\\_ACCESS\\_FLAG](#), only the [D3D11\\_BIND\\_SHADER\\_RESOURCE](#) and [D3D11\\_BIND\\_UNORDERED\\_ACCESS](#) [bind flags](#) may be used.

The [D3D11\\_RESOURCE\\_MISC\\_FLAG](#) cannot be used when creating resources with [D3D11\\_CPU\\_ACCESS](#) flags.

On non-unified memory architecture systems (discrete GPUs), apps should not use mappable default buffers if the compute shader code accesses the same byte in a default buffer more than once - sending the data across the bus multiple times eliminates the performance gained by mapping the default buffer instead of copying it.

# Requirements

Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

[Core Structures](#)

[D3D11\\_FEATURE](#)

---

## Feedback

Was this page helpful?



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

# D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS2 structure (d3d11.h)

Article07/27/2022

Describes Direct3D 11.3 feature options in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D11_OPTIONS2 {
    BOOL PSSpecifiedStencilRefSupported;
    BOOL TypedUAVLoadAdditionalFormats;
    BOOL ROVsSupported;
    D3D11_CONSERVATIVE_RASTERIZATION_TIER ConservativeRasterizationTier;
    D3D11_TILED_RESOURCES_TIER TiledResourcesTier;
    BOOL MapOnDefaultTextures;
    BOOL StandardSwizzle;
    BOOL UnifiedMemoryArchitecture;
} D3D11_FEATURE_DATA_D3D11_OPTIONS2;
```

## Members

PSSpecifiedStencilRefSupported

Specifies whether the hardware and driver support **PSSpecifiedStencilRef**. The runtime sets this member to **TRUE** if the hardware and driver support this option.

TypedUAVLoadAdditionalFormats

Specifies whether the hardware and driver support **TypedUAVLoadAdditionalFormats**. The runtime sets this member to **TRUE** if the hardware and driver support this option.

ROVsSupported

Specifies whether the hardware and driver support ROVs. The runtime sets this member to **TRUE** if the hardware and driver support this option.

ConservativeRasterizationTier

Specifies whether the hardware and driver support conservative rasterization. The runtime sets this member to a **D3D11\_CONSERVATIVE\_RASTERIZATION\_TIER**-typed

value that indicates if the hardware and driver support conservative rasterization and at what tier level.

#### TiledResourcesTier

Specifies whether the hardware and driver support tiled resources. The runtime sets this member to a [D3D11\\_TILED\\_RESOURCES\\_TIER](#)-typed value that indicates if the hardware and driver support tiled resources and at what tier level.

#### MapOnDefaultTextures

Specifies whether the hardware and driver support mapping on default textures. The runtime sets this member to **TRUE** if the hardware and driver support this option.

#### StandardSwizzle

Specifies whether the hardware and driver support standard swizzle. The runtime sets this member to **TRUE** if the hardware and driver support this option.

#### UnifiedMemoryArchitecture

Specifies whether the hardware and driver support Unified Memory Architecture. The runtime sets this member to **TRUE** if the hardware and driver support this option.

## Remarks

If **MapOnDefaultTextures** is **TRUE**, applications may create textures using **D3D11\_USAGE\_DEFAULT** in combination with non-zero a **D3D11\_CPU\_ACCESS\_FLAG** value. For performance reasons it is typically undesirable to create a default texture with CPU access flags unless the **UnifiedMemoryArchitecture** option is **TRUE**, or CPU / GPU usage of the texture is tightly interleaved.

Default textures may not be in a mapped state while either bound to the pipeline to referenced by an operation issued to a context. Default textures may not be mapped by a deferred context. Default textures may not be created shareable.

See [D3D11\\_TEXTURE\\_LAYOUT](#) for texture swizzle options and restrictions.

## Requirements

Minimum supported client	Windows 10 [desktop apps only]
--------------------------	--------------------------------

Minimum supported server	Windows Server 2016 [desktop apps only]
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?



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

# D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS3 structure (d3d11.h)

Article 02/22/2024

Describes Direct3D 11.3 feature options in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D11_OPTIONS3 {
    BOOL VPAndRTArrayIndexFromAnyShaderFeedingRasterizer;
} D3D11_FEATURE_DATA_D3D11_OPTIONS3;
```

## Members

VPAndRTArrayIndexFromAnyShaderFeedingRasterizer

Whether to use the VP and RT array index from any shader feeding the rasterizer.

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

[D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#)

[D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS1](#)

[D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS4 structure (d3d11\_4.h)

Article02/22/2024

Describes Direct3D 11.4 feature options in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D11_OPTIONS4 {
    BOOL ExtendedNV12SharedTextureSupported;
} D3D11_FEATURE_DATA_D3D11_OPTIONS4;
```

## Members

`ExtendedNV12SharedTextureSupported`

Specifies a BOOL that determines if NV12 textures can be shared across processes and D3D devices.

## Remarks

Use this structure with the D3D11\_FEATURE\_D3D11\_OPTIONS4 member of [D3D11\\_FEATURE](#).

Refer to the section on NV12 in [Direct3D 11.4 Features](#).

## Requirements

 Expand table

Requirement	Value
Header	d3d11_4.h

## See also

[Core Structures](#)

# D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS5 structure (d3d11.h)

Article02/22/2024

Describes the level of support for shared resources in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_D3D11_OPTIONS5 {
    D3D11_SHARED_RESOURCE_TIER SharedResourceTier;
} D3D11_FEATURE_DATA_D3D11_OPTIONS5;
```

## Members

SharedResourceTier

Type: [D3D11\\_SHARED\\_RESOURCE\\_TIER](#)

The level of support for shared resources in the current graphics driver.

## Remarks

Use this structure with the `D3D11_FEATURE_D3D11_OPTIONS5` member of [D3D11\\_FEATURE](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d11.h

## See also

[Direct3D 11 core structures](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_FEATURE\_DATA\_DISPLAYABLE structure (d3d11.h)

Article02/22/2024

Describes the level of displayable surfaces supported in the current graphics driver. See [Displayable surfaces](#).

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_DISPLAYABLE {
    BOOL             DisplayableTexture;
    D3D11_SHARED_RESOURCE_TIER SharedResourceTier;
} D3D11_FEATURE_DATA_DISPLAYABLE;
```

## Members

`DisplayableTexture`

Type: [BOOL](#)

`true` if the driver supports displayable surfaces; otherwise, `false`.

`SharedResourceTier`

Type: [D3D11\\_SHARED\\_RESOURCE\\_TIER](#)

The level of support for shared resources in the current graphics driver.

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 11 (Build 10.0.22000.194)
Minimum supported server	Windows 11 (Build 10.0.22000.194)
Header	d3d11.h

## See also

- [Displayable surfaces](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_DOUBLES structure (d3d11.h)

Article 02/22/2024

Describes double data type support in the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_DOUBLES {
    BOOL DoublePrecisionFloatShaderOps;
} D3D11_FEATURE_DATA_DOUBLES;
```

## Members

`DoublePrecisionFloatShaderOps`

Type: `BOOL`

Specifies whether `double` types are allowed. If `TRUE`, `double` types are allowed; otherwise `FALSE`. The runtime must set `DoublePrecisionFloatShaderOps` to `TRUE` in order for you to use any `HLSL` shader that is compiled with a `double` type.

## Remarks

If the runtime sets `DoublePrecisionFloatShaderOps` to `TRUE`, the hardware and driver support the following [Shader Model 5](#) instructions:

- `dadd`
- `dmax`
- `dmin`
- `dmul`
- `deq`
- `dge`
- `dlt`
- `dne`
- `dmov`
- `dmovc`

- [dtof](#)
- [ftod](#)

**Note** If DoublePrecisionFloatShaderOps is TRUE, the hardware and driver do not necessarily support double-precision division.

## Requirements

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

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_FORMAT\_SUPPORT structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_FORMAT_SUPPORT {
    DXGI_FORMAT InFormat;
    UINT         OutFormatSupport;
} D3D11_FEATURE_DATA_FORMAT_SUPPORT;
```

## Members

`InFormat`

Type: [DXGI\\_FORMAT](#)

[DXGI\\_FORMAT](#) to return information on.

`OutFormatSupport`

Type: [UINT](#)

Combination of [D3D11\\_FORMAT\\_SUPPORT](#) flags indicating which resources are supported.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_FORMAT\_SUPPORT2 structure (d3d11.h)

Article 02/22/2024

Describes which unordered resource options are supported by the current graphics driver for a given format.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_FORMAT_SUPPORT2 {
    DXGI_FORMAT InFormat;
    UINT         OutFormatSupport2;
} D3D11_FEATURE_DATA_FORMAT_SUPPORT2;
```

## Members

InFormat

Type: [DXGI\\_FORMAT](#)

[DXGI\\_FORMAT](#) to return information on.

OutFormatSupport2

Type: [UINT](#)

Combination of [D3D11\\_FORMAT\\_SUPPORT2](#) flags indicating which unordered resource options are supported.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_GPU\_VIRTUAL\_ADDRESS\_SUPPORT structure (d3d11.h)

Article 02/22/2024

Describes feature data GPU virtual address support, including maximum address bits per resource and per process.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT {
    UINT MaxGPUVirtualAddressBitsPerResource;
    UINT MaxGPUVirtualAddressBitsPerProcess;
} D3D11_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT;
```

## Members

`MaxGPUVirtualAddressBitsPerResource`

The maximum GPU virtual address bits per resource.

`MaxGPUVirtualAddressBitsPerProcess`

The maximum GPU virtual address bits per process.

## Remarks

See [D3D11\\_FEATURE](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_MARKER\_SUPPORT structure (d3d11.h)

Article02/22/2024

**Note** This structure is supported by the Direct3D 11.2 runtime, which is available on Windows 8.1 and later operating systems.

Describes whether a GPU profiling technique is supported.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_MARKER_SUPPORT {
    BOOL Profile;
} D3D11_FEATURE_DATA_MARKER_SUPPORT;
```

## Members

### Profile

Specifies whether the hardware and driver support a GPU profiling technique that can be used with development tools. The runtime sets this member to **TRUE** if the hardware and driver support data marking.

## Remarks

If the Direct3D API is the Direct3D 11.2 runtime and can support 11.2 features, [ID3D11Device::CheckFeatureSupport](#) for **D3D11\_FEATURE\_MARKER\_SUPPORT** will return a **SUCCESS** code when valid parameters are passed. The **Profile** member of **D3D11\_FEATURE\_DATA\_MARKER\_SUPPORT** will be set to **TRUE** or **FALSE**.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

[Core Structures](#)

[D3D11\\_FEATURE](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_SHADER\_CACHE structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_SHADER_CACHE {
    UINT SupportFlags;
} D3D11_FEATURE_DATA_SHADER_CACHE;
```

## Members

`SupportFlags`

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 D3D11\_FEATURE\_SHADER\_CACHE in the [D3D11\\_FEATURE](#) enumeration.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h (include D3d11_3.h)

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_SHADER\_MIN\_PRECISION\_SUPPORT structure (d3d11.h)

Article04/02/2021

**Note** This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

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

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_SHADER_MIN_PRECISION_SUPPORT {
    UINT PixelShaderMinPrecision;
    UINT AllOtherShaderStagesMinPrecision;
} D3D11_FEATURE_DATA_SHADER_MIN_PRECISION_SUPPORT;
```

## Members

`PixelShaderMinPrecision`

A combination of [D3D11\\_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 the pixel shader. A value of zero indicates that the driver supports only full 32-bit precision for the pixel shader.

`AllOtherShaderStagesMinPrecision`

A combination of [D3D11\\_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 all other shader stages. A value of zero indicates that the driver supports only full 32-bit precision for all other shader stages.

## Remarks

For hardware at Direct3D 10 and higher [feature levels](#), the runtime sets both members identically. For hardware at Direct3D 9.3 and lower feature levels, the runtime can set a

lower precision support in the **PixelShaderMinPrecision** member than the **AllOtherShaderStagesMinPrecision** member; for 9.3 and lower, all other shader stages represent only the vertex shader.

For more info about HLSL minimum precision, see [using HLSL minimum precision](#).

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

[Core Structures](#)

[D3D11\\_FEATURE](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE\_DATA\_THREADING structure (d3d11.h)

Article 02/22/2024

Describes the multi-threading features that are supported by the current graphics driver.

## Syntax

C++

```
typedef struct D3D11_FEATURE_DATA_THREADING {
    BOOL DriverConcurrentCreates;
    BOOL DriverCommandLists;
} D3D11_FEATURE_DATA_THREADING;
```

## Members

`DriverConcurrentCreates`

Type: [BOOL](#)

**TRUE** means resources can be created concurrently on multiple threads while drawing;  
**FALSE** means that the presence of coarse synchronization will prevent concurrency.

`DriverCommandLists`

Type: [BOOL](#)

**TRUE** means command lists are supported by the current driver; **FALSE** means that the API will emulate deferred contexts and command lists with software.

## Remarks

Use the `D3D11_FEATURE_DATA_THREADING` structure with the [`ID3D11Device::CheckFeatureSupport`](#) method to determine multi-threading support.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_INPUT\_ELEMENT\_DESC structure (d3d11.h)

Article 07/27/2022

A description of a single element for the input-assembler stage.

## Syntax

C++

```
typedef struct D3D11_INPUT_ELEMENT_DESC {
    LPCSTR SemanticName;
    UINT SemanticIndex;
    DXGI_FORMAT Format;
    UINT InputSlot;
    UINT AlignedByteOffset;
    D3D11_INPUT_CLASSIFICATION InputSlotClass;
    UINT InstanceDataStepRate;
} D3D11_INPUT_ELEMENT_DESC;
```

## Members

SemanticName

Type: [LPCSTR](#)

The HLSL semantic associated with this element in a shader input-signature. See [HLSL Semantics](#) for more info.

SemanticIndex

Type: [UINT](#)

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

Type: [DXGI\\_FORMAT](#)

The data type of the element data. See [DXGI\\_FORMAT](#).

#### InputSlot

Type: [UINT](#)

An integer value that identifies the input-assembler (see input slot). Valid values are between 0 and 15, defined in D3D11.h.

#### AlignedByteOffset

Type: [UINT](#)

Optional. Offset (in bytes) from the start of the vertex. Use D3D11\_APPEND\_ALIGNED\_ELEMENT for convenience to define the current element directly after the previous one, including any packing if necessary.

#### InputSlotClass

Type: [D3D11\\_INPUT\\_CLASSIFICATION](#)

Identifies the input data class for a single input slot (see [D3D11\\_INPUT\\_CLASSIFICATION](#)).

#### InstanceDataStepRate

Type: [UINT](#)

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 D3D11\_INPUT\_PER\_VERTEX\_DATA).

## Remarks

An input-layout object contains an array of structures, each structure defines one element being read from an input slot. Create an input-layout object by calling [ID3D11Device::CreateInputLayout](#). For an example, see the "Create the Input-Layout Object" subtopic under the [Getting Started with the Input-Assembler Stage](#) topic.

# Requirements

Header	d3d11.h
--------	---------

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_QUERY\_DATA\_PIPELINE\_STATISTICS structure (d3d11.h)

Article04/02/2021

Query information about graphics-pipeline activity in between calls to [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#).

## Syntax

C++

```
typedef struct D3D11_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;
} D3D11_QUERY_DATA_PIPELINE_STATISTICS;
```

## Members

`IAVertices`

Type: [UINT64](#)

Number of vertices read by input assembler.

`IAPrimitives`

Type: [UINT64](#)

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.

`VSIInvocations`

Type: **UINT64**

Number of times a vertex shader was invoked. Direct3D invokes the vertex shader once per vertex.

**GSI**nvocations

Type: **UINT64**

Number of times a geometry shader was invoked. When the geometry shader is set to **NULL**, this statistic may or may not increment depending on the hardware manufacturer.

**GSP**rimitives

Type: **UINT64**

Number of primitives output by a geometry shader.

**C**I nvocations

Type: **UINT64**

Number of primitives that were sent to the rasterizer. When the rasterizer is disabled, this will not increment.

**CP**rimitives

Type: **UINT64**

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.

**PSI**nvocations

Type: **UINT64**

Number of times a pixel shader was invoked.

**HSI**nvocations

Type: **UINT64**

Number of times a hull shader was invoked.

**DSI**nvocations

Type: **UINT64**

Number of times a domain shader was invoked.

CSInvocations

Type: [UINT64](#)

Number of times a compute shader was invoked.

## Requirements

Header	d3d11.h
--------	---------

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_QUERY\_DATA\_SO\_STATISTICS structure (d3d11.h)

Article 02/22/2024

Query information about the amount of data streamed out to the stream-output buffers in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#).

## Syntax

C++

```
typedef struct D3D11_QUERY_DATA_SO_STATISTICS {
    UINT64 NumPrimitivesWritten;
    UINT64 PrimitivesStorageNeeded;
} D3D11_QUERY_DATA_SO_STATISTICS;
```

## Members

`NumPrimitivesWritten`

Type: [UINT64](#)

Number of primitives (that is, points, lines, and triangles) written to the stream-output buffers.

`PrimitivesStorageNeeded`

Type: [UINT64](#)

Number of primitives that would have been written to the stream-output buffers if there had been enough space for them all.

## Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_QUERY\_DATA\_TIMESTAMP\_DISJOINT structure (d3d11.h)

Article 02/22/2024

Query information about the reliability of a timestamp query.

## Syntax

C++

```
typedef struct D3D11_QUERY_DATA_TIMESTAMP_DISJOINT {
    UINT64 Frequency;
    BOOL   Disjoint;
} D3D11_QUERY_DATA_TIMESTAMP_DISJOINT;
```

## Members

Frequency

Type: [UINT64](#)

How frequently the GPU counter increments in Hz.

Disjoint

Type: [BOOL](#)

If this is [TRUE](#), something occurred in between the query's [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#) calls that caused the timestamp counter to become discontinuous or disjoint, such as unplugging the AC cord on a laptop, overheating, or throttling up/down due to laptop savings events. The timestamp returned by [ID3D11DeviceContext::GetData](#) for a timestamp query is only reliable if [Disjoint](#) is [FALSE](#).

## Remarks

For a list of query types see [D3D11\\_QUERY](#).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_QUERY\_DESC structure (d3d11.h)

Article 02/22/2024

Describes a query.

## Syntax

C++

```
typedef struct D3D11_QUERY_DESC {
    D3D11_QUERY Query;
    UINT         MiscFlags;
} D3D11_QUERY_DESC;
```

## Members

Query

Type: [D3D11\\_QUERY](#)

Type of query (see [D3D11\\_QUERY](#)).

MiscFlags

Type: [UINT](#)

Miscellaneous flags (see [D3D11\\_QUERY\\_MISC\\_FLAG](#)).

## Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# CD3D11\_QUERY\_DESC1 structure (d3d11\_3.h)

Article 02/22/2024

Describes a query.

## Syntax

C++

```
struct CD3D11_QUERY_DESC1 : D3D11_QUERY_DESC1 {
    void CD3D11_QUERY_DESC1();
    void CD3D11_QUERY_DESC1(
        const D3D11_QUERY_DESC1 & o
    );
    void CD3D11_QUERY_DESC1(
        D3D11_QUERY         query,
        UINT                miscFlags,
        D3D11_CONTEXT_TYPE contextType
    );
    void ~CD3D11_QUERY_DESC1();
};
```

## Inheritance

The CD3D11\_QUERY\_DESC1 structure implements D3D11\_QUERY\_DESC1.

## Members

void CD3D11\_QUERY\_DESC1()

TBD

void CD3D11\_QUERY\_DESC1( const D3D11\_QUERY\_DESC1 & o)

void CD3D11\_QUERY\_DESC1( D3D11\_QUERY query, UINT miscFlags, D3D11\_CONTEXT\_TYPE contextType)

void ~CD3D11\_QUERY\_DESC1()

TBD

# Requirements

 Expand table

Requirement	Value
Header	d3d11_3.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_RASTERIZER\_DESC structure (d3d11.h)

Article 07/27/2022

Describes rasterizer state.

## Syntax

C++

```
typedef struct D3D11_RASTERIZER_DESC {
    D3D11_FILL_MODE FillMode;
    D3D11_CULL_MODE CullMode;
    BOOL             FrontCounterClockwise;
    INT              DepthBias;
    FLOAT            DepthBiasClamp;
    FLOAT            SlopeScaledDepthBias;
    BOOL             DepthClipEnable;
    BOOL             ScissorEnable;
    BOOL             MultisampleEnable;
    BOOL             AntialiasedLineEnable;
} D3D11_RASTERIZER_DESC;
```

## Members

FillMode

Type: [D3D11\\_FILL\\_MODE](#)

Determines the fill mode to use when rendering (see [D3D11\\_FILL\\_MODE](#)).

CullMode

Type: [D3D11\\_CULL\\_MODE](#)

Indicates triangles facing the specified direction are not drawn (see [D3D11\\_CULL\\_MODE](#)).

FrontCounterClockwise

Type: [BOOL](#)

Determines if a triangle is front- or back-facing. If this parameter 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`

Type: **INT**

Depth value added to a given pixel. For info about depth bias, see [Depth Bias](#).

`DepthBiasClamp`

Type: **FLOAT**

Maximum depth bias of a pixel. For info about depth bias, see [Depth Bias](#).

`SlopeScaledDepthBias`

Type: **FLOAT**

Scalar on a given pixel's slope. For info about depth bias, see [Depth Bias](#).

`DepthClipEnable`

Type: **BOOL**

Enable clipping based on distance.

The hardware always performs x and y clipping of rasterized coordinates. When **DepthClipEnable** is set to the default—**TRUE**, the hardware also clips the z value (that is, the hardware performs the last step of the following algorithm).

syntax

```
0 < w  
-w ≤ x ≤ w (or arbitrarily wider range if implementation uses a  
guard band to reduce clipping burden)  
-w ≤ y ≤ w (or arbitrarily wider range if implementation uses a  
guard band to reduce clipping burden)  
0 ≤ z ≤ w
```

When you set **DepthClipEnable** to **FALSE**, the hardware skips the z clipping (that is, the last step in the preceding algorithm). However, the hardware still performs the " $0 < w$ " clipping. When z clipping is disabled, improper depth ordering at the pixel level might

result. However, when z clipping is disabled, stencil shadow implementations are simplified. In other words, you can avoid complex special-case handling for geometry that goes beyond the back clipping plane.

#### ScissorEnable

Type: **BOOL**

Enable scissor-rectangle culling. All pixels outside an active scissor rectangle are culled.

#### MultisampleEnable

Type: **BOOL**

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

Type: **BOOL**

Specifies whether to enable line antialiasing; only applies if doing line drawing and **MultisampleEnable** is **FALSE**. For more info about this member, see Remarks.

## Remarks

Rasterizer state defines the behavior of the rasterizer stage. To create a rasterizer-state object, call [ID3D11Device::CreateRasterizerState](#). To set rasterizer state, call [ID3D11DeviceContext::RSSetState](#).

If you do not specify some rasterizer state, the Direct3D runtime uses the following default values for rasterizer state.

State	Default Value
FillMode	Solid
CullMode	Back
FrontCounterClockwise	<b>FALSE</b>
DepthBias	0
SlopeScaledDepthBias	0.0f

DepthBiasClamp	0.0f
DepthClipEnable	TRUE
ScissorEnable	FALSE
MultisampleEnable	FALSE
AntialiasedLineEnable	FALSE

**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	d3d11.h
--------	---------

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# CD3D11\_RASTERIZER\_DESC1 structure (d3d11\_1.h)

Article02/16/2023

**Note** This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

Describes rasterizer state.

## Syntax

C++

```
struct CD3D11_RASTERIZER_DESC1 : D3D11_RASTERIZER_DESC1 {
    void CD3D11_RASTERIZER_DESC1();
    void CD3D11_RASTERIZER_DESC1(
        const D3D11_RASTERIZER_DESC1 & o
    );
    void CD3D11_RASTERIZER_DESC1(
        CD3D11_DEFAULT unnamedParam1
    );
    void CD3D11_RASTERIZER_DESC1(
        D3D11_FILL_MODE fillMode,
        D3D11_CULL_MODE cullMode,
        BOOL frontCounterClockwise,
        INT depthBias,
        FLOAT depthBiasClamp,
        FLOAT slopeScaledDepthBias,
        BOOL depthClipEnable,
        BOOL scissorEnable,
        BOOL multisampleEnable,
        BOOL antialiasedLineEnable,
        UINT forcedSampleCount
    );
    void ~CD3D11_RASTERIZER_DESC1();
};
```

## Inheritance

The **CD3D11\_RASTERIZER\_DESC1** structure implements **D3D11\_RASTERIZER\_DESC1**.

# Members

```
void CD3D11_RASTERIZER_DESC1()

void CD3D11_RASTERIZER_DESC1( const D3D11_RASTERIZER_DESC1 & o)

void CD3D11_RASTERIZER_DESC1( CD3D11_DEFAULT unnamedParam1)

void CD3D11_RASTERIZER_DESC1( D3D11_FILL_MODE fillMode, D3D11_CULL_MODE cullMode,
BOOL frontCounterClockwise, INT depthBias, FLOAT depthBiasClamp, FLOAT
slopeScaledDepthBias, BOOL depthClipEnable, BOOL scissorEnable, BOOL
multisampleEnable, BOOL antialiasedLineEnable, UINT forcedSampleCount)

void ~CD3D11_RASTERIZER_DESC1()
```

## Remarks

AntialiasedLineEnable

Type: [BOOL](#)

Specifies whether to enable line antialiasing; only applies if doing line drawing and **MultisampleEnable** is **FALSE**. For more info about this member, see Remarks.

CullMode

Type: [D3D11\\_CULL\\_MODE](#)

Indicates that triangles facing the specified direction are not drawn.

DepthBias

Type: [INT](#)

Depth value added to a given pixel. For info about depth bias, see [Depth Bias](#).

DepthBiasClamp

Type: [FLOAT](#)

Maximum depth bias of a pixel. For info about depth bias, see [Depth Bias](#).

DepthClipEnable

Type: [BOOL](#)

Specifies whether to enable clipping based on distance.

The hardware always performs x and y clipping of rasterized coordinates. When **DepthClipEnable** is set to the default—TRUE, the hardware also clips the z value (that is, the hardware performs the last step of the following algorithm).

#### syntax

```
0 < w  
-w <= x <= w (or arbitrarily wider range if implementation uses a  
guard band to reduce clipping burden)  
-w <= y <= w (or arbitrarily wider range if implementation uses a  
guard band to reduce clipping burden)  
0 <= z <= w
```

When you set **DepthClipEnable** to FALSE, the hardware skips the z clipping (that is, the last step in the preceding algorithm). However, the hardware still performs the "0 < w" clipping. When z clipping is disabled, improper depth ordering at the pixel level might result. However, when z clipping is disabled, stencil shadow implementations are simplified. In other words, you can avoid complex special-case handling for geometry that goes beyond the back clipping plane.

#### FillMode

Type: [D3D11\\_FILL\\_MODE](#)

Determines the fill mode to use when rendering.

#### 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 ([D3D11\\_BIND\\_RENDER\\_TARGET](#)) 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, `ID3D11ShaderReflection::IsSampleFrequencyShader` returns **FALSE**.

Otherwise, rendering behavior is undefined. For info about how to configure depth-stencil, see [Configuring Depth-Stencil Functionality](#).

## FrontCounterClockwise

Type: **BOOL**

Specifies whether a triangle is front- or back-facing. If **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 **FALSE**, the opposite is true.

## MultisampleEnable

Type: **BOOL**

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.

## ScissorEnable

Type: **BOOL**

Specifies whether to enable scissor-rectangle culling. All pixels outside an active scissor rectangle are culled.

## SlopeScaledDepthBias

Type: **FLOAT**

Scalar on a given pixel's slope. For info about depth bias, see [Depth Bias](#).

Rasterizer state defines the behavior of the rasterizer stage. To create a rasterizer-state object, call `ID3D11Device1::CreateRasterizerState1`. To set rasterizer state, call `ID3D11DeviceContext::RSSetState`.

If you do not specify some rasterizer state, the Direct3D runtime uses the following default values for rasterizer state.

State	Default Value
-------	---------------

<b>FillMode</b>	Solid
<b>CullMode</b>	Back
<b>FrontCounterClockwise</b>	FALSE
<b>DepthBias</b>	0
<b>SlopeScaledDepthBias</b>	0.0f
<b>DepthBiasClamp</b>	0.0f
<b>DepthClipEnable</b>	TRUE
<b>ScissorEnable</b>	FALSE
<b>MultisampleEnable</b>	FALSE
<b>AntialiasedLineEnable</b>	FALSE
<b>ForcedSampleCount</b>	0

**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:

<b>Line-rendering algorithm</b>	<b>MultisampleEnable</b>	<b>AntialiasedLineEnable</b>
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

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11_1.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?



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

# CD3D11\_RASTERIZER\_DESC2 structure (d3d11\_3.h)

Article 02/16/2023

Describes rasterizer state.

## Syntax

C++

```
struct CD3D11_RASTERIZER_DESC2 : D3D11_RASTERIZER_DESC2 {
    void CD3D11_RASTERIZER_DESC2();
    void CD3D11_RASTERIZER_DESC2(
        const D3D11_RASTERIZER_DESC2 & o
    );
    void CD3D11_RASTERIZER_DESC2(
        CD3D11_DEFAULT unnamedParam1
    );
    void CD3D11_RASTERIZER_DESC2(
        D3D11_FILL_MODE fillMode,
        D3D11_CULL_MODE cullMode,
        BOOL frontCounterClockwise,
        INT depthBias,
        FLOAT depthBiasClamp,
        FLOAT slopeScaledDepthBias,
        BOOL depthClipEnable,
        BOOL scissorEnable,
        BOOL multisampleEnable,
        BOOL antialiasedLineEnable,
        UINT forcedSampleCount,
        D3D11_CONSERVATIVE_RASTERIZATION_MODE conservativeRaster
    );
    void ~CD3D11_RASTERIZER_DESC2();
};
```

## Inheritance

The CD3D11\_RASTERIZER\_DESC2 structure implements D3D11\_RASTERIZER\_DESC2.

## Members

```
void CD3D11_RASTERIZER_DESC2()
```

TBD

```

void CD3D11_RASTERIZER_DESC2( const D3D11_RASTERIZER_DESC2 & o)

void CD3D11_RASTERIZER_DESC2( CD3D11_DEFAULT unnamedParam1)

void CD3D11_RASTERIZER_DESC2( D3D11_FILL_MODE fillMode, D3D11_CULL_MODE cullMode,
BOOL frontCounterClockwise, INT depthBias, FLOAT depthBiasClamp, FLOAT
slopeScaledDepthBias, BOOL depthClipEnable, BOOL scissorEnable, BOOL
multisampleEnable, BOOL antialiasedLineEnable, UINT forcedSampleCount,
D3D11_CONSERVATIVE_RASTERIZATION_MODE conservativeRaster)

void ~CD3D11_RASTERIZER_DESC2()

```

TBD

## Remarks

Rasterizer state defines the behavior of the rasterizer stage. To create a rasterizer-state object, call [ID3D11Device3::CreateRasterizerState2](#). To set rasterizer state, call [ID3D11DeviceContext::RSSetState](#).

If you do not specify some rasterizer state, the Direct3D runtime uses the following default values for rasterizer state.

State	Default Value
FillMode	Solid
CullMode	Back
FrontCounterClockwise	FALSE
DepthBias	0
SlopeScaledDepthBias	0.0f
DepthBiasClamp	0.0f
DepthClipEnable	TRUE
ScissorEnable	FALSE
MultisampleEnable	FALSE
AntialiasedLineEnable	FALSE
ForcedSampleCount	0
ConservativeRaster	D3D11_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	d3d11_3.h
--------	-----------

## See also

[Core Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3D11\_RECT

Article • 11/19/2022

D3D11\_RECT is declared as follows:

syntax

```
typedef RECT D3D11_RECT;
```

For more information about this GDI rectangle structure, see [RECT](#).

## Remarks

This structure is used for scissor rectangles by [ID3D11DeviceContext::RSGetScissorRects](#) and [ID3D11DeviceContext::RSSetScissorRects](#).

## Requirements

Requirement	Value
Header	D3D11.h
Library	D3D11.lib

## See also

[Core Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_RENDER\_TARGET\_BLEND\_DESC structure (d3d11.h)

Article 07/27/2022

Describes the blend state for a render target.

## Syntax

C++

```
typedef struct D3D11_RENDER_TARGET_BLEND_DESC {
    BOOL          BlendEnable;
    D3D11_BLEND   SrcBlend;
    D3D11_BLEND   DestBlend;
    D3D11_BLEND_OP BlendOp;
    D3D11_BLEND   SrcBlendAlpha;
    D3D11_BLEND   DestBlendAlpha;
    D3D11_BLEND_OP BlendOpAlpha;
    UINT8         RenderTargetWriteMask;
} D3D11_RENDER_TARGET_BLEND_DESC;
```

## Members

**BlendEnable**

Type: **BOOL**

Enable (or disable) blending.

**SrcBlend**

Type: **D3D11\_BLEND**

This **blend option** 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**

Type: **D3D11\_BLEND**

This **blend option** 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**

Type: [D3D11\\_BLEND\\_OP](#)

This [blend operation](#) defines how to combine the **SrcBlend** and **DestBlend** operations.

**SrcBlendAlpha**

Type: [D3D11\\_BLEND](#)

This [blend option](#) 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**

Type: [D3D11\\_BLEND](#)

This [blend option](#) 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**

Type: [D3D11\\_BLEND\\_OP](#)

This [blend operation](#) defines how to combine the **SrcBlendAlpha** and **DestBlendAlpha** operations.

**RenderTargetWriteMask**

Type: [UINT8](#)

A write mask.

## Remarks

You specify an array of [D3D11\\_RENDER\\_TARGET\\_BLEND\\_DESC](#) structures in the **RenderTarget** member of the [D3D11\\_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
SrcBlend	D3D11_BLEND_ONE
DestBlend	D3D11_BLEND_ZERO
BlendOp	D3D11_BLEND_OP_ADD
SrcBlendAlpha	D3D11_BLEND_ONE
DestBlendAlpha	D3D11_BLEND_ZERO
BlendOpAlpha	D3D11_BLEND_OP_ADD
RenderTargetWriteMask	D3D11_COLOR_WRITE_ENABLE_ALL

## Requirements

Header	d3d11.h
--------	---------

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_RENDER\_TARGET\_BLEND\_DESC1 structure (d3d11\_1.h)

Article 07/27/2022

Describes the blend state for a render target.

## ⓘ Note

This structure is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

## Syntax

C++

```
typedef struct D3D11_RENDER_TARGET_BLEND_DESC1 {
    BOOL          BlendEnable;
    BOOL          LogicOpEnable;
    D3D11_BLEND   SrcBlend;
    D3D11_BLEND   DestBlend;
    D3D11_BLEND_OP BlendOp;
    D3D11_BLEND   SrcBlendAlpha;
    D3D11_BLEND   DestBlendAlpha;
    D3D11_BLEND_OP BlendOpAlpha;
    D3D11_LOGIC_OP LogicOp;
    UINT8         RenderTargetWriteMask;
} D3D11_RENDER_TARGET_BLEND_DESC1;
```

## Members

**BlendEnable**

Type: **BOOL**

Enable (or disable) blending.

## ⓘ Note

It's not valid for *LogicOpEnable* and *BlendEnable* to both be **TRUE**.

### LogicOpEnable

Type: **BOOL**

Enable (or disable) a logical operation.

#### ⓘ Note

If you set *LogicOpEnable* to **TRUE**, then *BlendEnable* must be **FALSE**, and the system's **D3D11\_FEATURE\_DATA\_D3D11\_OPTIONS::OutputMergerLogicOp** option must be **TRUE**.

### SrcBlend

Type: **D3D11\_BLEND**

This [blend option](#) 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

Type: **D3D11\_BLEND**

This [blend option](#) 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

Type: **D3D11\_BLEND\_OP**

This [blend operation](#) defines how to combine the **SrcBlend** and **DestBlend** operations.

### SrcBlendAlpha

Type: **D3D11\_BLEND**

This [blend option](#) 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

Type: **D3D11\_BLEND**

This [blend option](#) 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**

Type: [D3D11\\_BLEND\\_OP](#)

This [blend operation](#) defines how to combine the **SrcBlendAlpha** and **DestBlendAlpha** operations.

**LogicOp**

Type: [D3D11\\_LOGIC\\_OP](#)

A [D3D11\\_LOGIC\\_OP](#)-typed value that specifies the logical operation to configure for the render target.

**RenderTargetWriteMask**

Type: [UINT8](#)

A write mask.

## Remarks

### ⓘ Note

It's not valid for *LogicOpEnable* and *BlendEnable* to both be **TRUE**.

You specify an array of [D3D11\\_RENDER\\_TARGET\\_BLEND\\_DESC1](#) structures in the **RenderTarget** member of the [D3D11\\_BLEND\\_DESC1](#) 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	D3D11_BLEND_ONE
DestBlend	D3D11_BLEND_ZERO
BlendOp	D3D11_BLEND_OP_ADD
SrcBlendAlpha	D3D11_BLEND_ONE
DestBlendAlpha	D3D11_BLEND_ZERO
BlendOpAlpha	D3D11_BLEND_OP_ADD
LogicOp	D3D11_LOGIC_OP_NOOP
RenderTargetWriteMask	D3D11_COLOR_WRITE_ENABLE_ALL

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11_1.h

## See also

[Core Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SAMPLER\_DESC structure (d3d11.h)

Article 09/01/2022

Describes a sampler state.

## Syntax

C++

```
typedef struct D3D11_SAMPLER_DESC {
    D3D11_FILTER             Filter;
    D3D11_TEXTURE_ADDRESS_MODE AddressU;
    D3D11_TEXTURE_ADDRESS_MODE AddressV;
    D3D11_TEXTURE_ADDRESS_MODE AddressW;
    FLOAT                    MipLODBias;
    UINT                     MaxAnisotropy;
    D3D11_COMPARISON_FUNC    ComparisonFunc;
    FLOAT                    BorderColor[4];
    FLOAT                    MinLOD;
    FLOAT                    MaxLOD;
} D3D11_SAMPLER_DESC;
```

## Members

Filter

Type: [D3D11\\_FILTER](#)

Filtering method to use when sampling a texture (see [D3D11\\_FILTER](#)).

AddressU

Type: [D3D11\\_TEXTURE\\_ADDRESS\\_MODE](#)

Method to use for resolving a u texture coordinate that is outside the 0 to 1 range (see [D3D11\\_TEXTURE\\_ADDRESS\\_MODE](#)).

AddressV

Type: [D3D11\\_TEXTURE\\_ADDRESS\\_MODE](#)

Method to use for resolving a v texture coordinate that is outside the 0 to 1 range.

`AddressW`

Type: [D3D11\\_TEXTURE\\_ADDRESS\\_MODE](#)

Method to use for resolving a w texture coordinate that is outside the 0 to 1 range.

`MipLODBias`

Type: [FLOAT](#)

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`

Type: [UINT](#)

Clamping value used if D3D11\_FILTER\_ANISOTROPIC or D3D11\_FILTER\_COMPARISON\_ANISOTROPIC is specified in Filter. Valid values are between 1 and 16.

`ComparisonFunc`

Type: [D3D11\\_COMPARISON\\_FUNC](#)

A function that compares sampled data against existing sampled data. The function options are listed in [D3D11\\_COMPARISON\\_FUNC](#).

`BorderColor[4]`

Type: [FLOAT\[4\]](#)

Border color to use if D3D11\_TEXTURE\_ADDRESS\_BORDER is specified for AddressU, AddressV, or AddressW. Range must be between 0.0 and 1.0 inclusive.

`MinLOD`

Type: [FLOAT](#)

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`

Type: [FLOAT](#)

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 D3D11\_FLOAT32\_MAX.

## Remarks

These are the default values for sampler state.

State	Default Value
Filter	D3D11_FILTER_MIN_MAG_MIP_LINEAR
AddressU	D3D11_TEXTURE_ADDRESS_CLAMP
AddressV	D3D11_TEXTURE_ADDRESS_CLAMP
AddressW	D3D11_TEXTURE_ADDRESS_CLAMP
MinLOD	-3.402823466e+38F (-FLT_MAX)
MaxLOD	3.402823466e+38F (FLT_MAX)
MipMapLODBias	0.0f
MaxAnisotropy	1
ComparisonFunc	D3D11_COMPARISON_NEVER
BorderColor	float4(1.0f,1.0f,1.0f,1.0f)
Texture	N/A

## Requirements

Header	d3d11.h
--------	---------

## See also

[Core Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3D11\_SO\_DECLARATION\_ENTRY structure (d3d11.h)

Article 02/22/2024

Description of a vertex element in a vertex buffer in an output slot.

## Syntax

C++

```
typedef struct D3D11_SO_DECLARATION_ENTRY {
    UINT    Stream;
    LPCSTR SemanticName;
    UINT    SemanticIndex;
    BYTE    StartComponent;
    BYTE    ComponentCount;
    BYTE    OutputSlot;
} D3D11_SO_DECLARATION_ENTRY;
```

## Members

Stream

Type: [UINT](#)

Zero-based, stream number.

SemanticName

Type: [LPCSTR](#)

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

Type: [UINT](#)

Output element's zero-based index. Should be used if, for example, you have more than one texture coordinate stored in each vertex.

`StartComponent`

Type: **BYTE**

Which 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, then `StartComponent` should be 1 and `ComponentCount` should be 2.

`ComponentCount`

Type: **BYTE**

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, then `StartComponent` should be 1 and `ComponentCount` should be 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`

Type: **BYTE**

The associated stream output buffer that is bound to the pipeline (see [ID3D11DeviceContext::SOSetTargets](#)). The valid range for `OutputSlot` is 0 to 3.

## Requirements

[ ] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

## Feedback

Was this page helpful?

 Yes

 No



# D3D11\_VIEWPORT structure (d3d11.h)

Article 07/27/2022

Defines the dimensions of a viewport.

## Syntax

C++

```
typedef struct D3D11_VIEWPORT {  
    FLOAT TopLeftX;  
    FLOAT TopLeftY;  
    FLOAT Width;  
    FLOAT Height;  
    FLOAT MinDepth;  
    FLOAT MaxDepth;  
} D3D11_VIEWPORT;
```

## Members

`TopLeftX`

Type: [FLOAT](#)

X position of the left hand side of the viewport. Ranges between `D3D11_VIEWPORT_BOUNDS_MIN` and `D3D11_VIEWPORT_BOUNDS_MAX`.

`TopLeftY`

Type: [FLOAT](#)

Y position of the top of the viewport. Ranges between `D3D11_VIEWPORT_BOUNDS_MIN` and `D3D11_VIEWPORT_BOUNDS_MAX`.

`Width`

Type: [FLOAT](#)

Width of the viewport.

`Height`

Type: [FLOAT](#)

Height of the viewport.

MinDepth

Type: **FLOAT**

Minimum depth of the viewport. Ranges between 0 and 1.

MaxDepth

Type: **FLOAT**

Maximum depth of the viewport. Ranges between 0 and 1.

## Remarks

In all cases, **Width** and **Height** must be  $\geq 0$  and **TopLeftX + Width** and **TopLeftY + Height** must be  $\leq \text{D3D11_VIEWPORT_BOUNDS_MAX}$ .

Viewport Sizes and Feature Level Support Differences between Direct3D 11 and Direct3D 10:  
The range for the minimum and maximum viewport size is dependent on the feature level defined by [D3D\\_FEATURE\\_LEVEL](#).

- Direct3D 11 supports fractional viewports; the parameter types are floating-point numbers. The feature level, D3D\_FEATURE\_LEVEL\_11\_0, supports (D3D11\_VIEWPORT\_BOUNDS\_MIN, D3D11\_VIEWPORT\_BOUNDS\_MAX) values between (-32768, 32,767).
- Direct3D 10 does not support fractional viewports. The feature levels, D3D\_FEATURE\_LEVEL\_10\_1 (or below), supports (D3D10\_VIEWPORT\_BOUNDS\_MIN, D3D10\_VIEWPORT\_BOUNDS\_MAX) values between (-16384, 16383).

**Note** Even though you specify float values to the members of the **D3D11\_VIEWPORT** structure for the *pViewports* array in a call to **ID3D11DeviceContext::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.

## Requirements

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Direct3D 11 core enumerations

Article • 07/06/2022

This section contains information about the core enumerations.

## In this section

Topic	Description
<a href="#">D3D11_ASYNC_GETDATA_FLAG</a>	Optional flags that control the behavior of <a href="#">ID3D11DeviceContext::GetData</a> .
<a href="#">D3D11_BLEND</a>	Blend factors, which modulate values for the pixel shader and render target.
<a href="#">D3D11_BLEND_OP</a>	RGB or alpha blending operation.
<a href="#">D3D11_CLEAR_FLAG</a>	Specifies the parts of the depth stencil to clear.
<a href="#">D3D11_COLOR_WRITE_ENABLE</a>	Identify which components of each pixel of a render target are writable during blending.
<a href="#">D3D11_COMPARISON_FUNC</a>	Comparison options.
<a href="#">D3D11_CONSERVATIVE_RASTERIZATION_MODE</a>	Identifies whether conservative rasterization is on or off.
<a href="#">D3D11_CONSERVATIVE_RASTERIZATION_TIER</a>	Specifies if the hardware and driver support conservative rasterization and at what tier level.
<a href="#">D3D11_CONTEXT_TYPE</a>	Specifies the context in which a query occurs.
<a href="#">D3D11_COPY_FLAGS</a>	<p>[!Note]</p> <p>This enumeration is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p>
<a href="#">D3D11_COUNTER</a>	Specifies how to handle the existing contents of a resource during a copy or update operation of a region within that resource.
	Options for performance counters.

Topic	Description
<a href="#">D3D11_COUNTER_TYPE</a>	Data type of a performance counter.
<a href="#">D3D11_CREATE_DEVICE_FLAG</a>	Describes parameters that are used to create a device.
<a href="#">D3D11_1_CREATE_DEVICE_CONTEXT_STATE_FLAG</a>	Describes flags that are used to create a device context state object ( <a href="#">ID3DDeviceContextState</a> ) with the <a href="#">ID3D11Device1::CreateDeviceContextState</a> method.
<a href="#">D3D11_CULL_MODE</a>	Indicates triangles facing a particular direction are not drawn.
<a href="#">D3D11_DEPTH_WRITE_MASK</a>	Identify the portion of a depth-stencil buffer for writing depth data.
<a href="#">D3D11_DEVICE_CONTEXT_TYPE</a>	Device context options.
<a href="#">D3D11_FEATURE</a>	Direct3D 11 feature options.
<a href="#">D3D11_FENCE_FLAG</a>	Specifies fence options.
<a href="#">D3D11_FILL_MODE</a>	Determines the fill mode to use when rendering triangles.
<a href="#">D3D11_FILTER</a>	Filtering options during texture sampling.
<a href="#">D3D11_FILTER_TYPE</a>	Types of magnification or minification sampler filters.
<a href="#">D3D11_FILTER_REDUCTION_TYPE</a>	Specifies the type of sampler filter reduction.
<a href="#">D3D11_FORMAT_SUPPORT</a>	Which resources are supported for a given format and given device (see <a href="#">ID3D11Device::CheckFormatSupport</a> and <a href="#">ID3D11Device::CheckFeatureSupport</a> ).
<a href="#">D3D11_FORMAT_SUPPORT2</a>	Unordered resource support options for a compute shader resource (see <a href="#">ID3D11Device::CheckFeatureSupport</a> ).
<a href="#">D3D11_INPUT_CLASSIFICATION</a>	Type of data contained in an input slot.

Topic	Description
<a href="#">D3D11_LOGIC_OP</a>	<p>[!Note]</p> <p>This enumeration is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p> <p>Specifies logical operations to configure for a render target.</p>
<a href="#">D3D11_PRIMITIVE</a>	Indicates how the pipeline interprets geometry or hull shader input primitives.
<a href="#">D3D11_PRIMITIVE_TOPOLOGY</a>	How the pipeline interprets vertex data that is bound to the input-assembler stage. These primitive topology values determine how the vertex data is rendered on screen.
<a href="#">D3D11_QUERY</a>	Query types.
<a href="#">D3D11_QUERY_MISC_FLAG</a>	Flags that describe miscellaneous query behavior.
<a href="#">D3D11_RAISE_FLAG</a>	Option(s) for raising an error to a non-continuable exception.
<a href="#">D3D11_SHADER_CACHE_SUPPORT_FLAGS</a>	Describes the level of support for shader caching in the current graphics driver.
<a href="#">D3D11_SHADER_MIN_PRECISION_SUPPORT</a>	<p>[!Note]</p> <p>This enumeration is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.</p> <p>Values that specify minimum precision levels at shader stages.</p>
<a href="#">D3D11_SHARED_RESOURCE_TIER</a>	Defines constants that specify the level of support for shared resources in the current graphics driver.
<a href="#">D3D11_STENCIL_OP</a>	The stencil operations that can be performed during depth-stencil testing.

Topic	Description
D3D11_TEXTURE_ADDRESS_MODE	Identify a technique for resolving texture coordinates that are outside of the boundaries of a texture.
D3D11_TEXTURECUBE_FACE	The different faces of a cube texture.
D3D11_TILED_RESOURCES_TIER	Indicates the tier level at which tiled resources are supported.

## Related topics

[Core reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_ASYNC\_GETDATA\_FLAG enumeration (d3d11.h)

Article 02/22/2024

Optional flags that control the behavior of [ID3D11DeviceContext::GetData](#).

## Syntax

C++

```
typedef enum D3D11_ASYNC_GETDATA_FLAG {
    D3D11_ASYNC_GETDATA_DONOTFLUSH = 0x1
} ;
```

## Constants

[+] [Expand table](#)

`D3D11_ASYNC_GETDATA_DONOTFLUSH`

Value: `0x1`

Do not flush the command buffer. This can potentially cause an infinite loop if GetData is continually called until it returns `S_OK` as there may still be commands in the command buffer that need to be processed in order for GetData to return `S_OK`. Since the commands in the command buffer are not flushed they will not be processed and therefore GetData will never return `S_OK`.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BLEND enumeration (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef enum D3D11_BLEND {
    D3D11_BLEND_ZERO = 1,
    D3D11_BLEND_ONE = 2,
    D3D11_BLEND_SRC_COLOR = 3,
    D3D11_BLEND_INV_SRC_COLOR = 4,
    D3D11_BLEND_SRC_ALPHA = 5,
    D3D11_BLEND_INV_SRC_ALPHA = 6,
    D3D11_BLEND_DEST_ALPHA = 7,
    D3D11_BLEND_INV_DEST_ALPHA = 8,
    D3D11_BLEND_DEST_COLOR = 9,
    D3D11_BLEND_INV_DEST_COLOR = 10,
    D3D11_BLEND_SRC_ALPHA_SAT = 11,
    D3D11_BLEND_BLEND_FACTOR = 14,
    D3D11_BLEND_INV_BLEND_FACTOR = 15,
    D3D11_BLEND_SRC1_COLOR = 16,
    D3D11_BLEND_INV_SRC1_COLOR = 17,
    D3D11_BLEND_SRC1_ALPHA = 18,
    D3D11_BLEND_INV_SRC1_ALPHA = 19
} ;
```

## Constants

[Expand table](#)

D3D11\_BLEND\_ZERO

Value: 1

The blend factor is (0, 0, 0, 0). No pre-blend operation.

D3D11\_BLEND\_ONE

Value: 2

The blend factor is (1, 1, 1, 1). No pre-blend operation.

D3D11\_BLEND\_SRC\_COLOR

Value: 3

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

D3D11\_BLEND\_INV\_SRC\_COLOR

Value: 4

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

D3D11\_BLEND\_SRC\_ALPHA

Value: 5

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

D3D11\_BLEND\_INV\_SRC\_ALPHA

Value: 6

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

D3D11\_BLEND\_DEST\_ALPHA

Value: 7

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

D3D11\_BLEND\_INV\_DEST\_ALPHA

Value: 8

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

D3D11\_BLEND\_DEST\_COLOR

Value: 9

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

D3D11\_BLEND\_INV\_DEST\_COLOR

Value: 10

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

D3D11\_BLEND\_SRC\_ALPHA\_SAT

Value: 11

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

D3D11\_BLEND\_BLEND\_FACTOR

Value: 14

The blend factor is the blend factor set with [ID3D11DeviceContext::OMSetBlendState](#). No pre-blend operation.

D3D11\_BLEND\_INV\_BLEND\_FACTOR

Value: 15

The blend factor is the blend factor set with [ID3D11DeviceContext::OMSetBlendState](#). The pre-blend operation inverts the blend factor, generating  $1 - \text{blend\_factor}$ .

D3D11\_BLEND\_SRC1\_COLOR

Value: 16

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

D3D11\_BLEND\_INV\_SRC1\_COLOR

Value: 17

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

D3D11\_BLEND\_SRC1\_ALPHA

Value: 18

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

D3D11\_BLEND\_INV\_SRC1\_ALPHA

Value: 19

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

## Remarks

Blend operations are specified in a [blend description](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BLEND\_OP enumeration (d3d11.h)

Article 07/27/2022

RGB or alpha blending operation.

## Syntax

C++

```
typedef enum D3D11_BLEND_OP {
    D3D11_BLEND_OP_ADD = 1,
    D3D11_BLEND_OP_SUBTRACT = 2,
    D3D11_BLEND_OP_REV_SUBTRACT = 3,
    D3D11_BLEND_OP_MIN = 4,
    D3D11_BLEND_OP_MAX = 5
} ;
```

## Constants

`D3D11_BLEND_OP_ADD`

Value: 1

Add source 1 and source 2.

`D3D11_BLEND_OP_SUBTRACT`

Value: 2

Subtract source 1 from source 2.

`D3D11_BLEND_OP_REV_SUBTRACT`

Value: 3

Subtract source 2 from source 1.

`D3D11_BLEND_OP_MIN`

Value: 4

Find the minimum of source 1 and source 2.

`D3D11_BLEND_OP_MAX`

Value: 5

Find the maximum of source 1 and source 2.

## Remarks

The runtime implements RGB blending and alpha blending separately. Therefore, blend state requires separate blend operations for RGB data and alpha data. These blend operations are specified in a [blend description](#). 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 [blend option](#) controls the data source that the blending stage uses to modulate values for the pixel shader, render target, or both. The [blend operation](#) controls how the blending stage mathematically combines these modulated values.

## Requirements

Header	d3d11.h
--------	---------

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_CLEAR\_FLAG enumeration (d3d11.h)

Article02/22/2024

Specifies the parts of the depth stencil to clear.

## Syntax

C++

```
typedef enum D3D11_CLEAR_FLAG {
    D3D11_CLEAR_DEPTH = 0x1L,
    D3D11_CLEAR_STENCIL = 0x2L
} ;
```

## Constants

[+] Expand table

`D3D11_CLEAR_DEPTH`

Value: `0x1L`

Clear the depth buffer, using fast clear if possible, then place the resource in a compressed state.

`D3D11_CLEAR_STENCIL`

Value: `0x2L`

Clear the stencil buffer, using fast clear if possible, then place the resource in a compressed state.

## Remarks

These flags are used when calling [ID3D11DeviceContext::ClearDepthStencilView](#); the flags can be combined with a bitwise OR.

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?



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

# D3D11\_COLOR\_WRITE\_ENABLE enumeration (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef enum D3D11_COLOR_WRITE_ENABLE {
    D3D11_COLOR_WRITE_ENABLE_RED = 1,
    D3D11_COLOR_WRITE_ENABLE_GREEN = 2,
    D3D11_COLOR_WRITE_ENABLE_BLUE = 4,
    D3D11_COLOR_WRITE_ENABLE_ALPHA = 8,
    D3D11_COLOR_WRITE_ENABLE_ALL
};
```

## Constants

[] Expand table

	<code>D3D11_COLOR_WRITE_ENABLE_RED</code>
Value:	7
Allow data to be stored in the red component.	
	<code>D3D11_COLOR_WRITE_ENABLE_GREEN</code>
Value:	2
Allow data to be stored in the green component.	
	<code>D3D11_COLOR_WRITE_ENABLE_BLUE</code>
Value:	4
Allow data to be stored in the blue component.	
	<code>D3D11_COLOR_WRITE_ENABLE_ALPHA</code>
Value:	8
Allow data to be stored in the alpha component.	
	<code>D3D11_COLOR_WRITE_ENABLE_ALL</code>
Allow data to be stored in all components.	

## Remarks

These flags can be combined with a bitwise OR.

## Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_COMPARISON\_FUNC enumeration (d3d11.h)

Article 07/27/2022

Comparison options.

## Syntax

C++

```
typedef enum D3D11_COMPARISON_FUNC {
    D3D11_COMPARISON_NEVER = 1,
    D3D11_COMPARISON_LESS = 2,
    D3D11_COMPARISON_EQUAL = 3,
    D3D11_COMPARISON_LESS_EQUAL = 4,
    D3D11_COMPARISON_GREATER = 5,
    D3D11_COMPARISON_NOT_EQUAL = 6,
    D3D11_COMPARISON_GREATER_EQUAL = 7,
    D3D11_COMPARISON_ALWAYS = 8
} ;
```

## Constants

`D3D11_COMPARISON_NEVER`

Value: 1

Never pass the comparison.

`D3D11_COMPARISON_LESS`

Value: 2

If the source data is less than the destination data, the comparison passes.

`D3D11_COMPARISON_EQUAL`

Value: 3

If the source data is equal to the destination data, the comparison passes.

`D3D11_COMPARISON_LESS_EQUAL`

Value: 4

If the source data is less than or equal to the destination data, the comparison passes.

`D3D11_COMPARISON_GREATER`

Value: 5

If the source data is greater than the destination data, the comparison passes.

`D3D11_COMPARISON_NOT_EQUAL`

Value: 6

If the source data is not equal to the destination data, the comparison passes.

`D3D11_COMPARISON_GREATER_EQUAL`

Value: 7

If the source data is greater than or equal to the destination data, the comparison passes.

`D3D11_COMPARISON_ALWAYS`

Value: 8

Always pass the comparison.

## Remarks

A comparison option determines whether 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 (see [D3D11\\_DEPTH\\_STENCIL\\_DESC](#)), depth-stencil operations (see [D3D11\\_DEPTH\\_STENCILOP\\_DESC](#)), or sampler state (see [D3D11\\_SAMPLER\\_DESC](#)).

## Requirements

Header

d3d11.h

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_CONSERVATIVE\_RASTERIZATION\_MODE enumeration (d3d11\_3.h)

Article 02/22/2024

Identifies whether conservative rasterization is on or off.

## Syntax

C++

```
typedef enum D3D11_CONSERVATIVE_RASTERIZATION_MODE {
    D3D11_CONSERVATIVE_RASTERIZATION_MODE_OFF = 0,
    D3D11_CONSERVATIVE_RASTERIZATION_MODE_ON = 1
};
```

## Constants

  Expand table

<code>D3D11_CONSERVATIVE_RASTERIZATION_MODE_OFF</code>	Value: 0 Conservative rasterization is off.
<code>D3D11_CONSERVATIVE_RASTERIZATION_MODE_ON</code>	Value: 1 Conservative rasterization is on.

## Requirements

  Expand table

Requirement	Value
Header	d3d11_3.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_CONSERVATIVE\_RASTERIZATION\_TIER enumeration (d3d11.h)

Article 02/22/2024

Specifies if the hardware and driver support conservative rasterization and at what tier level.

## Syntax

C++

```
typedef enum D3D11_CONSERVATIVE_RASTERIZATION_TIER {
    D3D11_CONSERVATIVE_RASTERIZATION_NOT_SUPPORTED = 0,
    D3D11_CONSERVATIVE_RASTERIZATION_TIER_1 = 1,
    D3D11_CONSERVATIVE_RASTERIZATION_TIER_2 = 2,
    D3D11_CONSERVATIVE_RASTERIZATION_TIER_3 = 3
};
```

## Constants

[Expand table](#)

<b>D3D11_CONSERVATIVE_RASTERIZATION_NOT_SUPPORTED</b> Value: 0 Conservative rasterization isn't supported.
<b>D3D11_CONSERVATIVE_RASTERIZATION_TIER_1</b> Value: 1 Tier_1 conservative rasterization is supported.
<b>D3D11_CONSERVATIVE_RASTERIZATION_TIER_2</b> Value: 2 Tier_2 conservative rasterization is supported.
<b>D3D11_CONSERVATIVE_RASTERIZATION_TIER_3</b> Value: 3 Tier_3 conservative rasterization is supported.

## Requirements

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_CONTEXT\_TYPE enumeration (d3d11\_3.h)

Article02/22/2024

Specifies the context in which a query occurs.

## Syntax

C++

```
typedef enum D3D11_CONTEXT_TYPE {
    D3D11_CONTEXT_TYPE_ALL = 0,
    D3D11_CONTEXT_TYPE_3D = 1,
    D3D11_CONTEXT_TYPE_COMPUTE = 2,
    D3D11_CONTEXT_TYPE_COPY = 3,
    D3D11_CONTEXT_TYPE_VIDEO = 4
} ;
```

## Constants

[+] Expand table

<b>D3D11_CONTEXT_TYPE_ALL</b> Value: 0 The query can occur in all contexts.
<b>D3D11_CONTEXT_TYPE_3D</b> Value: 1 The query occurs in the context of a 3D command queue.
<b>D3D11_CONTEXT_TYPE_COMPUTE</b> Value: 2 The query occurs in the context of a 3D compute queue.
<b>D3D11_CONTEXT_TYPE_COPY</b> Value: 3 The query occurs in the context of a 3D copy queue.
<b>D3D11_CONTEXT_TYPE_VIDEO</b> Value: 4 The query occurs in the context of video.

# Remarks

This enum is used by the following:

- [D3D11\\_QUERY\\_DESC1](#) structure
- A [CD3D11\\_QUERY\\_DESC1](#) constructor.
- [ID3D11DeviceContext3::Flush1](#) method

# Requirements

  [Expand table](#)

Requirement	Value
Header	d3d11_3.h

## See also

[Core Enumerations](#)

[D3D11\\_QUERY\\_DESC1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_COPY\_FLAGS enumeration (d3d11\_1.h)

Article02/22/2024

**Note** This enumeration is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

Specifies how to handle the existing contents of a resource during a copy or update operation of a region within that resource.

## Syntax

C++

```
typedef enum D3D11_COPY_FLAGS {
    D3D11_COPY_NO_OVERWRITE = 0x1,
    D3D11_COPY_DISCARD = 0x2
};
```

## Constants

[+] Expand table

`D3D11_COPY_NO_OVERWRITE`

Value: `0x1`

The existing contents of the resource cannot be overwritten.

`D3D11_COPY_DISCARD`

Value: `0x2`

The existing contents of the resource are undefined and can be discarded.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Header	d3d11_1.h

## See also

[Core Enumerations](#)

[ID3D11DeviceContext1::CopySubresourceRegion1](#)

[ID3D11DeviceContext1::UpdateSubresource1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_COUNTER enumeration (d3d11.h)

Article02/22/2024

Options for performance counters.

## Syntax

C++

```
typedef enum D3D11_COUNTER {
    D3D11_COUNTER_DEVICE_DEPENDENT_0 = 0x40000000
} ;
```

## Constants

[+] Expand table

`D3D11_COUNTER_DEVICE_DEPENDENT_0`

Value: `0x40000000`

Define a performance counter that is dependent on the hardware device.

## Remarks

Independent hardware vendors may define their own set of performance counters for their devices, by giving the enumeration value a number that is greater than the value for `D3D11_COUNTER_DEVICE_DEPENDENT_0`.

This enumeration is used by [D3D11\\_COUNTER\\_DESC](#) and [D3D11\\_COUNTER\\_INFO](#).

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_COUNTER\_TYPE enumeration (d3d11.h)

Article02/22/2024

Data type of a performance counter.

## Syntax

C++

```
typedef enum D3D11_COUNTER_TYPE {
    D3D11_COUNTER_TYPE_FLOAT32 = 0,
    D3D11_COUNTER_TYPE_UINT16,
    D3D11_COUNTER_TYPE_UINT32,
    D3D11_COUNTER_TYPE_UINT64
};
```

## Constants

[+] Expand table

`D3D11_COUNTER_TYPE_FLOAT32`

Value: 0

32-bit floating point.

`D3D11_COUNTER_TYPE_UINT16`

16-bit unsigned integer.

`D3D11_COUNTER_TYPE_UINT32`

32-bit unsigned integer.

`D3D11_COUNTER_TYPE_UINT64`

64-bit unsigned integer.

## Remarks

These flags are an output parameter in [ID3D11Device::CheckCounter](#).

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_CREATE\_DEVICE\_FLAG enumeration (d3d11.h)

Article01/31/2022

Describes parameters that are used to create a device.

## Syntax

C++

```
typedef enum D3D11_CREATE_DEVICE_FLAG {
    D3D11_CREATE_DEVICE_SINGLETHREADED = 0x1,
    D3D11_CREATE_DEVICE_DEBUG = 0x2,
    D3D11_CREATE_DEVICE_SWITCH_TO_REF = 0x4,
    D3D11_CREATE_DEVICE_PREVENT_INTERNAL_THREADING_OPTIMIZATIONS = 0x8,
    D3D11_CREATE_DEVICE_BGRA_SUPPORT = 0x20,
    D3D11_CREATE_DEVICE_DEBUGGABLE = 0x40,
    D3D11_CREATE_DEVICE_PREVENT_ALTERING_LAYER_SETTINGS_FROM_REGISTRY = 0x80,
    D3D11_CREATE_DEVICE_DISABLE_GPU_TIMEOUT = 0x100,
    D3D11_CREATE_DEVICE_VIDEO_SUPPORT = 0x800
} ;
```

## Constants

D3D11\_CREATE\_DEVICE\_SINGLETHREADED

Value: 0x1

Use this flag if your application will only call methods of Direct3D 11 interfaces from a single thread. By default, the [ID3D11Device](#) object is [thread-safe](#).

By using this flag, you can increase performance. However, if you use this flag and your application calls methods of Direct3D 11 interfaces from multiple threads, undefined behavior might result.

D3D11\_CREATE\_DEVICE\_DEBUG

Value: 0x2

Creates a device that supports the [debug layer](#).

To use this flag, you must have D3D11\*SDKLayers.dll installed; otherwise, device creation fails. To get D3D11\_1SDKLayers.dll, install the SDK for Windows 8.

`D3D11_CREATE_DEVICE_SWITCH_TO_REF`

Value: *0x4*

**Note** This flag is not supported in Direct3D 11.

`D3D11_CREATE_DEVICE_PREVENT_INTERNAL_THREADING_OPTIMIZATIONS`

Value: *0x8*

Prevents multiple threads from being created. When this flag is used with a [Windows Advanced Rasterization Platform \(WARP\)](#) device, no additional threads will be created by WARP and all rasterization will occur on the calling thread. This flag is not recommended for general use. See remarks.

`D3D11_CREATE_DEVICE_BGRA_SUPPORT`

Value: *0x20*

Creates a device that supports BGRA formats ([DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM](#) and [DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM\\_SRGB](#)). All 10level9 and higher hardware with WDDM 1.1+ drivers support BGRA formats.

**Note** Required for Direct2D interoperability with Direct3D resources.

`D3D11_CREATE_DEVICE_DEBUGGABLE`

Value: *0x40*

Causes the device and driver to keep information that you can use for shader debugging. The exact impact from this flag will vary from driver to driver.

To use this flag, you must have D3D11\_1SDKLayers.dll installed; otherwise, device creation fails. The created device supports the [debug layer](#). To get D3D11\_1SDKLayers.dll, install the SDK for Windows 8.

If you use this flag and the current driver does not support shader debugging, device creation fails. Shader debugging requires a driver that is implemented to the WDDM for Windows 8 (WDDM 1.2).

**Direct3D 11:** This value is not supported until Direct3D 11.1.

#### `D3D11_CREATE_DEVICE_PREVENT_ALTERING_LAYER_SETTINGS_FROM_REGISTRY`

Value: *0x80*

Causes the Direct3D runtime to ignore registry settings that turn on the [debug layer](#). You can turn on the debug layer by using the [DirectX Control Panel](#) that was included as part of the DirectX SDK. We shipped the last version of the DirectX SDK in June 2010; you can download it from the [Microsoft Download Center](#). You can set this flag in your app, typically in release builds only, to prevent end users from using the [DirectX Control Panel](#) to monitor how the app uses Direct3D.

**Note** You can also set this flag in your app to prevent Direct3D debugging tools, such as Visual Studio Ultimate 2012, from hooking your app.

**Windows 8.1:** This flag doesn't prevent Visual Studio 2013 and later running on Windows 8.1 and later from hooking your app; instead use [ID3D11DeviceContext2::IsAnnotationEnabled](#). This flag still prevents Visual Studio 2013 and later running on Windows 8 and earlier from hooking your app.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

#### `D3D11_CREATE_DEVICE_DISABLE_GPU_TIMEOUT`

Value: *0x100*

Use this flag if the device will produce GPU workloads that take more than two seconds to complete, and you want the operating system to allow them to successfully finish. If this flag is not set, the operating system performs [timeout detection and recovery](#) when it detects a GPU packet that took more than two seconds to execute. If this flag is set, the operating system allows such a long running packet to execute without resetting the GPU. We recommend not to set this flag if your device needs to be highly responsive so that the operating system can detect and recover from GPU timeouts. We recommend to set this flag if your device needs to perform time consuming background tasks such as compute, image recognition, and video encoding to allow such tasks to successfully finish.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

#### `D3D11_CREATE_DEVICE_VIDEO_SUPPORT`

Value: `0x800`

Forces the creation of the Direct3D device to fail if the display driver is not implemented to the WDDM for Windows 8 (WDDM 1.2). When the display driver is not implemented to WDDM 1.2, only a Direct3D device that is created with [feature level](#) 9.1, 9.2, or 9.3 supports video; therefore, if this flag is set, the runtime creates the Direct3D device only for feature level 9.1, 9.2, or 9.3. We recommend not to specify this flag for applications that want to favor Direct3D capability over video. If feature level 10 and higher is available, the runtime will use that feature level regardless of video support.

If this flag is set, device creation on the Basic Render Device (BRD) will succeed regardless of the BRD's missing support for video decode. This is because the Media Foundation video stack operates in software mode on BRD. In this situation, if you force the video stack to create the Direct3D device twice (create the device once with this flag, next discover BRD, then again create the device without the flag), you actually degrade performance.

If you attempt to create a Direct3D device with driver type [D3D\\_DRIVER\\_TYPE\\_NULL](#), [D3D\\_DRIVER\\_TYPE\\_REFERENCE](#), or [D3D\\_DRIVER\\_TYPE\\_SOFTWARE](#), device creation fails at any [feature level](#) because none of the associated drivers provide video capability. If you attempt to create a Direct3D device with driver type [D3D\\_DRIVER\\_TYPE\\_WARP](#), device creation succeeds to allow software fallback for video.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

## Remarks

Device creation flags are used by [D3D11CreateDevice](#) and [D3D11CreateDeviceAndSwapChain](#).

An application might dynamically create (and destroy) threads to improve performance especially on a machine with multiple CPU cores. There may be cases, however, when an application needs to prevent extra threads from being created. This can happen when you want to simplify debugging, profile code or develop a tool for instance. For these cases, use [D3D11\\_CREATE\\_DEVICE\\_PREVENT\\_INTERNAL\\_THREADING\\_OPTIMIZATIONS](#) to request that the runtime and video driver not create any additional threads that might interfere with the application.

## Requirements

Header

d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_1\_CREATE\_DEVICE\_CONTEXT\_STATE\_FLAG enumeration (d3d11\_1.h)

Article 02/22/2024

Describes flags that are used to create a device context state object ([ID3DDeviceContextState](#)) with the [ID3D11Device1::CreateDeviceContextState](#) method.

## Syntax

C++

```
typedef enum D3D11_1_CREATE_DEVICE_CONTEXT_STATE_FLAG {
    D3D11_1_CREATE_DEVICE_CONTEXT_STATE_SINGLETHREADED = 0x1
} ;
```

## Constants

[+] [Expand table](#)

D3D11_1_CREATE_DEVICE_CONTEXT_STATE_SINGLETHREADED
--

Value: *0x1*

You use this flag if your application will only call methods of Direct3D 11 and Direct3D 10 interfaces from a single thread. By default, Direct3D 11 and Direct3D 10 are [thread-safe](#). By using this flag, you can increase performance. However, if you use this flag and your application calls methods from multiple threads, undefined behavior might result.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]

Requirement	Value
Header	d3d11_1.h

## See also

[Core Enumerations](#)

[ID3D11Device1::CreateDeviceContextState](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_CULL\_MODE enumeration (d3d11.h)

Article 02/22/2024

Indicates triangles facing a particular direction are not drawn.

## Syntax

C++

```
typedef enum D3D11_CULL_MODE {
    D3D11_CULL_NONE = 1,
    D3D11_CULL_FRONT = 2,
    D3D11_CULL_BACK = 3
};
```

## Constants

[+] Expand table

D3D11\_CULL\_NONE

Value: 1

Always draw all triangles.

D3D11\_CULL\_FRONT

Value: 2

Do not draw triangles that are front-facing.

D3D11\_CULL\_BACK

Value: 3

Do not draw triangles that are back-facing.

## Remarks

This enumeration is part of a rasterizer-state object description (see [D3D11\\_RASTERIZER\\_DESC](#)).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_DEPTH\_WRITE\_MASK enumeration (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef enum D3D11_DEPTH_WRITE_MASK {
    D3D11_DEPTH_WRITE_MASK_ZERO = 0,
    D3D11_DEPTH_WRITE_MASK_ALL = 1
};
```

## Constants

[] Expand table

<code>D3D11_DEPTH_WRITE_MASK_ZERO</code>	Value: 0 Turn off writes to the depth-stencil buffer.
<code>D3D11_DEPTH_WRITE_MASK_ALL</code>	Value: 1 Turn on writes to the depth-stencil buffer.

## Requirements

[] Expand table

Requirement	Value
Header	d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_DEVICE\_CONTEXT\_TYPE enumeration (d3d11.h)

Article 02/22/2024

Device context options.

## Syntax

C++

```
typedef enum D3D11_DEVICE_CONTEXT_TYPE {
    D3D11_DEVICE_CONTEXT_IMMEDIATE = 0,
    D3D11_DEVICE_CONTEXT_DEFERRED
} ;
```

## Constants

[+] Expand table

`D3D11_DEVICE_CONTEXT_IMMEDIATE`

Value: 0

The device context is an immediate context.

`D3D11_DEVICE_CONTEXT_DEFERRED`

The device context is a deferred context.

## Remarks

This enumeration is used by [ID3D11DeviceContext::GetType](#).

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FEATURE enumeration (d3d11.h)

Article07/07/2022

Direct3D 11 feature options.

## Syntax

C++

```
typedef enum D3D11_FEATURE {
    D3D11_FEATURE_THREADING = 0,
    D3D11_FEATURE_DOUBLES,
    D3D11_FEATURE_FORMAT_SUPPORT,
    D3D11_FEATURE_FORMAT_SUPPORT2,
    D3D11_FEATURE_D3D10_X_HARDWARE_OPTIONS,
    D3D11_FEATURE_D3D11_OPTIONS,
    D3D11_FEATURE_ARCHITECTURE_INFO,
    D3D11_FEATURE_D3D9_OPTIONS,
    D3D11_FEATURE_SHADER_MIN_PRECISION_SUPPORT,
    D3D11_FEATURE_D3D9_SHADOW_SUPPORT,
    D3D11_FEATURE_D3D11_OPTIONS1,
    D3D11_FEATURE_D3D9_SIMPLE_INSTANCING_SUPPORT,
    D3D11_FEATURE_MARKER_SUPPORT,
    D3D11_FEATURE_D3D9_OPTIONS1,
    D3D11_FEATURE_D3D11_OPTIONS2,
    D3D11_FEATURE_D3D11_OPTIONS3,
    D3D11_FEATURE_GPU_VIRTUAL_ADDRESS_SUPPORT,
    D3D11_FEATURE_D3D11_OPTIONS4,
    D3D11_FEATURE_SHADER_CACHE,
    D3D11_FEATURE_D3D11_OPTIONS5,
    D3D11_FEATURE_DISPLAYABLE
} ;
```

## Constants

`D3D11_FEATURE_THREADING`

Value: 0

The driver supports [multithreading](#). To see an example of testing a driver for multithread support, see [How To: Check for Driver Support](#). Refer to [D3D11\\_FEATURE\\_DATA\\_THREADING](#).

`D3D11_FEATURE_DOUBLES`

Supports the use of the double-precision shaders in HLSL. Refer to [D3D11\\_FEATURE\\_DATA\\_DOUBLES](#).

**D3D11\_FEATURE\_FORMAT\_SUPPORT**

Supports the formats in [D3D11\\_FORMAT\\_SUPPORT](#). Refer to [D3D11\\_FEATURE\\_DATA\\_FORMAT\\_SUPPORT](#).

**D3D11\_FEATURE\_FORMAT\_SUPPORT2**

Supports the formats in [D3D11\\_FORMAT\\_SUPPORT2](#). Refer to [D3D11\\_FEATURE\\_DATA\\_FORMAT\\_SUPPORT2](#).

**D3D11\_FEATURE\_D3D10\_X\_HARDWARE\_OPTIONS**

Supports compute shaders and raw and structured buffers. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D10\\_X\\_HARDWARE\\_OPTIONS](#).

**D3D11\_FEATURE\_D3D11\_OPTIONS**

Supports Direct3D 11.1 feature options. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#).

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_FEATURE\_ARCHITECTURE\_INFO**

Supports specific adapter architecture. Refer to [D3D11\\_FEATURE\\_DATA\\_ARCHITECTURE\\_INFO](#).

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_FEATURE\_D3D9\_OPTIONS**

Supports Direct3D 9 feature options. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D9\\_OPTIONS](#).

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_FEATURE\_SHADER\_MIN\_PRECISION\_SUPPORT**

Supports minimum precision of shaders. For more info about HLSL minimum precision, see [using HLSL minimum precision](#). Refer to [D3D11\\_FEATURE\\_DATA\\_SHADER\\_MIN\\_PRECISION\\_SUPPORT](#).

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_FEATURE\_D3D9\_SHADOW\_SUPPORT**

Supports Direct3D 9 shadowing feature. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D9\\_SHADOW\\_SUPPORT](#).

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_FEATURE\_D3D11\_OPTIONS1**

Supports Direct3D 11.2 feature options. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS1](#).

**Direct3D 11:** This value is not supported until Direct3D 11.2.

**D3D11\_FEATURE\_D3D9\_SIMPLE\_INSTANCING\_SUPPORT**

Supports Direct3D 11.2 instancing options. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D9\\_SIMPLE\\_INSTANCING\\_SUPPORT](#).

**Direct3D 11:** This value is not supported until Direct3D 11.2.

**D3D11\_FEATURE\_MARKER\_SUPPORT**

Supports Direct3D 11.2 marker options. Refer to [D3D11\\_FEATURE\\_DATA\\_MARKER\\_SUPPORT](#).

**Direct3D 11:** This value is not supported until Direct3D 11.2.

**D3D11\_FEATURE\_D3D9\_OPTIONS1**

Supports Direct3D 9 feature options, which includes the Direct3D 9 shadowing feature and instancing support. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D9\\_OPTIONS1](#).

**Direct3D 11:** This value is not supported until Direct3D 11.2.

**D3D11\_FEATURE\_D3D11\_OPTIONS2**

Supports Direct3D 11.3 conservative rasterization feature options. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#).

**Direct3D 11:** This value is not supported until Direct3D 11.3.

**D3D11\_FEATURE\_D3D11\_OPTIONS3**

Supports Direct3D 11.4 conservative rasterization feature options. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS3](#).

**Direct3D 11:** This value is not supported until Direct3D 11.4.

**D3D11\_FEATURE\_GPU\_VIRTUAL\_ADDRESS\_SUPPORT**

Supports GPU virtual addresses. Refer to [D3D11\\_FEATURE\\_DATA\\_GPU\\_VIRTUAL\\_ADDRESS\\_SUPPORT](#).

**D3D11\_FEATURE\_D3D11\_OPTIONS4**

Supports a single boolean for NV12 shared textures. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS4](#).

**Direct3D 11:** This value is not supported until Direct3D 11.4.

**D3D11\_FEATURE\_SHADER\_CACHE**

Supports shader cache, described in [D3D11\\_FEATURE\\_DATA\\_SHADER\\_CACHE](#).

**D3D11\_FEATURE\_D3D11\_OPTIONS5**

Supports a [D3D11\\_SHARED\\_RESOURCE\\_TIER](#) to indicate the level of support for shared resources in the current graphics driver. Refer to [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS5](#).

**D3D11\_FEATURE\_DISPLAYABLE**

Supports displayable surfaces, described in [D3D11\\_FEATURE\\_DATA\\_DISPLAYABLE](#).

## Remarks

This enumeration is used when querying a driver about support for these features by calling [ID3D11Device::CheckFeatureSupport](#). Each value in this enumeration has a corresponding data structure that is required to be passed to the *pFeatureSupportData* parameter of [ID3D11Device::CheckFeatureSupport](#).

## Requirements

Header	d3d11.h
--------	---------

## See also

[Core enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FENCE\_FLAG enumeration (d3d11\_3.h)

Article 02/22/2024

Specifies fence options.

## Syntax

C++

```
typedef enum D3D11_FENCE_FLAG {
    D3D11_FENCE_FLAG_NONE = 0,
    D3D11_FENCE_FLAG_SHARED = 0x2,
    D3D11_FENCE_FLAG_SHARED_CROSS_ADAPTER = 0x4,
    D3D11_FENCE_FLAG_NON_MONITORED = 0x8
} ;
```

## Constants

[ ] [Expand table](#)

`D3D11_FENCE_FLAG_NONE`

Value: *0*

No options are specified.

`D3D11_FENCE_FLAG_SHARED`

Value: *0x2*

The fence is shared.

`D3D11_FENCE_FLAG_SHARED_CROSS_ADAPTER`

Value: *0x4*

The fence is shared with another GPU adapter.

`D3D11_FENCE_FLAG_NON_MONITORED`

Value: *0x8*

## Remarks

This enum is used by the [ID3D11Device::CreateFence](#) method.

# Requirements

 Expand table

Requirement	Value
Header	d3d11_3.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FILL\_MODE enumeration (d3d11.h)

Article 02/22/2024

Determines the fill mode to use when rendering triangles.

## Syntax

C++

```
typedef enum D3D11_FILL_MODE {
    D3D11_FILL_WIREFRAME = 2,
    D3D11_FILL_SOLID = 3
} ;
```

## Constants

  Expand table

**D3D11\_FILL\_WIREFRAME**

Value: 2

Draw lines connecting the vertices. Adjacent vertices are not drawn.

**D3D11\_FILL\_SOLID**

Value: 3

Fill the triangles formed by the vertices. Adjacent vertices are not drawn.

## Remarks

This enumeration is part of a rasterizer-state object description (see [D3D11\\_RASTERIZER\\_DESC](#)).

## Requirements

  Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FILTER enumeration (d3d11.h)

Article07/27/2022

Filtering options during texture sampling.

## Syntax

C++

```
typedef enum D3D11_FILTER {
    D3D11_FILTER_MIN_MAG_MIP_POINT = 0,
    D3D11_FILTER_MIN_MAG_POINT_MIP_LINEAR = 0x1,
    D3D11_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT = 0x4,
    D3D11_FILTER_MIN_POINT_MAG_MIP_LINEAR = 0x5,
    D3D11_FILTER_MIN_LINEAR_MAG_MIP_POINT = 0x10,
    D3D11_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEAR = 0x11,
    D3D11_FILTER_MIN_MAG_LINEAR_MIP_POINT = 0x14,
    D3D11_FILTER_MIN_MAG_MIP_LINEAR = 0x15,
    D3D11_FILTER_ANISOTROPIC = 0x55,
    D3D11_FILTER_COMPARISON_MIN_MAG_MIP_POINT = 0x80,
    D3D11_FILTER_COMPARISON_MIN_MAG_POINT_MIP_LINEAR = 0x81,
    D3D11_FILTER_COMPARISON_MIN_POINT_MAG_LINEAR_MIP_POINT = 0x84,
    D3D11_FILTER_COMPARISON_MIN_POINT_MAG_MIP_LINEAR = 0x85,
    D3D11_FILTER_COMPARISON_MIN_LINEAR_MAG_MIP_POINT = 0x90,
    D3D11_FILTER_COMPARISON_MIN_LINEAR_MAG_POINT_MIP_LINEAR = 0x91,
    D3D11_FILTER_COMPARISON_MIN_MAG_LINEAR_MIP_POINT = 0x94,
    D3D11_FILTER_COMPARISON_MIN_MAG_MIP_LINEAR = 0x95,
    D3D11_FILTER_COMPARISON_ANISOTROPIC = 0xd5,
    D3D11_FILTER_MINIMUM_MIN_MAG_MIP_POINT = 0x100,
    D3D11_FILTER_MINIMUM_MIN_MAG_POINT_MIP_LINEAR = 0x101,
    D3D11_FILTER_MINIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT = 0x104,
    D3D11_FILTER_MINIMUM_MIN_POINT_MAG_MIP_LINEAR = 0x105,
    D3D11_FILTER_MINIMUM_MIN_LINEAR_MAG_MIP_POINT = 0x110,
    D3D11_FILTER_MINIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR = 0x111,
    D3D11_FILTER_MINIMUM_MIN_MAG_LINEAR_MIP_POINT = 0x114,
    D3D11_FILTER_MINIMUM_MIN_MAG_MIP_LINEAR = 0x115,
    D3D11_FILTER_MINIMUM_MIN_ANISOTROPIC = 0x155,
    D3D11_FILTER_MAXIMUM_MIN_MAG_MIP_POINT = 0x180,
    D3D11_FILTER_MAXIMUM_MIN_MAG_POINT_MIP_LINEAR = 0x181,
    D3D11_FILTER_MAXIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT = 0x184,
    D3D11_FILTER_MAXIMUM_MIN_POINT_MAG_MIP_LINEAR = 0x185,
    D3D11_FILTER_MAXIMUM_MIN_LINEAR_MAG_MIP_POINT = 0x190,
    D3D11_FILTER_MAXIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR = 0x191,
    D3D11_FILTER_MAXIMUM_MIN_MAG_LINEAR_MIP_POINT = 0x194,
    D3D11_FILTER_MAXIMUM_MIN_MAG_MIP_LINEAR = 0x195,
    D3D11_FILTER_MAXIMUM_ANISOTROPIC = 0x1d5
};
```

# Constants

D3D11\_FILTER\_MIN\_MAG\_MIP\_POINT

Value: 0

Use point sampling for minification, magnification, and mip-level sampling.

D3D11\_FILTER\_MIN\_MAG\_POINT\_MIP\_LINEAR

Value: 0x1

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

D3D11\_FILTER\_MIN\_POINT\_MAG\_LINEAR\_MIP\_POINT

Value: 0x4

Use point sampling for minification; use linear interpolation for magnification; use point sampling for mip-level sampling.

D3D11\_FILTER\_MIN\_POINT\_MAG\_MIP\_LINEAR

Value: 0x5

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

D3D11\_FILTER\_MIN\_LINEAR\_MAG\_MIP\_POINT

Value: 0x10

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

D3D11\_FILTER\_MIN\_LINEAR\_MAG\_POINT\_MIP\_LINEAR

Value: 0x11

Use linear interpolation for minification; use point sampling for magnification; use linear interpolation for mip-level sampling.

D3D11\_FILTER\_MIN\_MAG\_LINEAR\_MIP\_POINT

Value: 0x14

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

D3D11\_FILTER\_MIN\_MAG\_MIP\_LINEAR

Value: 0x15

Use linear interpolation for minification, magnification, and mip-level sampling.

D3D11\_FILTER\_ANISOTROPIC

Value: 0x55

Use anisotropic interpolation for minification, magnification, and mip-level sampling.

D3D11\_FILTER\_COMPARISON\_MIN\_MAG\_MIP\_POINT

Value: 0x80

Use point sampling for minification, magnification, and mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_COMPARISON\_MIN\_MAG\_POINT\_MIP\_LINEAR**

Value: 0x81

Use point sampling for minification and magnification; use linear interpolation for mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_COMPARISON\_MIN\_POINT\_MAG\_LINEAR\_MIP\_POINT**

Value: 0x84

Use point sampling for minification; use linear interpolation for magnification; use point sampling for mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_COMPARISON\_MIN\_POINT\_MAG\_MIP\_LINEAR**

Value: 0x85

Use point sampling for minification; use linear interpolation for magnification and mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_COMPARISON\_MIN\_LINEAR\_MAG\_MIP\_POINT**

Value: 0x90

Use linear interpolation for minification; use point sampling for magnification and mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_COMPARISON\_MIN\_LINEAR\_MAG\_POINT\_MIP\_LINEAR**

Value: 0x91

Use linear interpolation for minification; use point sampling for magnification; use linear interpolation for mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_COMPARISON\_MIN\_MAG\_LINEAR\_MIP\_POINT**

Value: 0x94

Use linear interpolation for minification and magnification; use point sampling for mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_COMPARISON\_MIN\_MAG\_MIP\_LINEAR**

Value: 0x95

Use linear interpolation for minification, magnification, and mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_COMPARISON\_ANISOTROPIC**

Value: 0xd5

Use anisotropic interpolation for minification, magnification, and mip-level sampling. Compare the result to the comparison value.

**D3D11\_FILTER\_MINIMUM\_MIN\_MAG\_MIP\_POINT**

Value: 0x100

Fetch the same set of texels as D3D11\_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.

`D3D11_FILTER_MINIMUM_MIN_MAG_POINT_MIP_LINEAR`

Value: `0x101`

Fetch the same set of texels as `D3D11_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.

`D3D11_FILTER_MINIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT`

Value: `0x104`

Fetch the same set of texels as `D3D11_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.

`D3D11_FILTER_MINIMUM_MIN_POINT_MAG_MIP_LINEAR`

Value: `0x105`

Fetch the same set of texels as `D3D11_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.

`D3D11_FILTER_MINIMUM_MIN_LINEAR_MAG_MIP_POINT`

Value: `0x110`

Fetch the same set of texels as `D3D11_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.

`D3D11_FILTER_MINIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR`

Value: `0x111`

Fetch the same set of texels as `D3D11_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.

`D3D11_FILTER_MINIMUM_MIN_MAG_LINEAR_MIP_POINT`

Value: `0x114`

Fetch the same set of texels as `D3D11_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.

`D3D11_FILTER_MINIMUM_MIN_MAG_MIP_LINEAR`

Value: `0x115`

Fetch the same set of texels as `D3D11_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.

**D3D11\_FILTER\_MINIMUM\_ANISOTROPIC**

Value: 0x155

Fetch the same set of texels as D3D11\_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.

**D3D11\_FILTER\_MAXIMUM\_MIN\_MAG\_MIP\_POINT**

Value: 0x180

Fetch the same set of texels as D3D11\_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.

**D3D11\_FILTER\_MAXIMUM\_MIN\_MAG\_POINT\_MIP\_LINEAR**

Value: 0x181

Fetch the same set of texels as D3D11\_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.

**D3D11\_FILTER\_MAXIMUM\_MIN\_POINT\_MAG\_LINEAR\_MIP\_POINT**

Value: 0x184

Fetch the same set of texels as D3D11\_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.

**D3D11\_FILTER\_MAXIMUM\_MIN\_POINT\_MAG\_MIP\_LINEAR**

Value: 0x185

Fetch the same set of texels as D3D11\_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.

**D3D11\_FILTER\_MAXIMUM\_MIN\_LINEAR\_MAG\_MIP\_POINT**

Value: 0x190

Fetch the same set of texels as D3D11\_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.

**D3D11\_FILTER\_MAXIMUM\_MIN\_LINEAR\_MAG\_POINT\_MIP\_LINEAR**

Value: 0x191

Fetch the same set of texels as D3D11\_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.

#### D3D11\_FILTER\_MAXIMUM\_MIN\_MAG\_LINEAR\_MIP\_POINT

Value: 0x794

Fetch the same set of texels as D3D11\_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.

#### D3D11\_FILTER\_MAXIMUM\_MIN\_MAG\_MIP\_LINEAR

Value: 0x195

Fetch the same set of texels as D3D11\_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.

#### D3D11\_FILTER\_MAXIMUM\_ANISOTROPIC

Value: 0x1d5

Fetch the same set of texels as D3D11\_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

**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.

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.

Texture Sampling Function	Texture Sampling Function with Comparison Filtering
sample	samplecmp or samplecmplevelzero

Comparison filters only work with textures that have the following DXGI formats:  
R32\_FLOAT\_X8X24\_TYPELESS, R32\_FLOAT, R24\_UNORM\_X8\_TYPELESS, R16\_UNORM.

## Requirements

Header	d3d11.h
--------	---------

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FILTER\_TYPE enumeration (d3d11.h)

Article07/27/2022

Types of magnification or minification sampler filters.

## Syntax

C++

```
typedef enum D3D11_FILTER_TYPE {
    D3D11_FILTER_TYPE_POINT = 0,
    D3D11_FILTER_TYPE_LINEAR = 1
};
```

## Constants

`D3D11_FILTER_TYPE_POINT`

Value: 0

Point filtering 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.

`D3D11_FILTER_TYPE_LINEAR`

Value: 1

Bilinear interpolation filtering 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.

## Requirements

Header

d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FILTER\_REDUCTION\_TYPE enumeration (d3d11.h)

Article02/22/2024

Specifies the type of sampler filter reduction.

## Syntax

C++

```
typedef enum D3D11_FILTER_REDUCTION_TYPE {
    D3D11_FILTER_REDUCTION_TYPE_STANDARD = 0,
    D3D11_FILTER_REDUCTION_TYPE_COMPARISON = 1,
    D3D11_FILTER_REDUCTION_TYPE_MINIMUM = 2,
    D3D11_FILTER_REDUCTION_TYPE_MAXIMUM = 3
};
```

## Constants

[ ] [Expand table](#)

<code>D3D11_FILTER_REDUCTION_TYPE_STANDARD</code>
---

Value: 0

Indicates standard (default) filter reduction.

<code>D3D11_FILTER_REDUCTION_TYPE_COMPARISON</code>
---

Value: 1

Indicates a comparison filter reduction.

<code>D3D11_FILTER_REDUCTION_TYPE_MINIMUM</code>
--

Value: 2

Indicates minimum filter reduction.

<code>D3D11_FILTER_REDUCTION_TYPE_MAXIMUM</code>
--

Value: 3

Indicates maximum filter reduction.

## Remarks

This enum is used by the [D3D11\\_SAMPLER\\_DESC](#) structure.

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FORMAT\_SUPPORT enumeration (d3d11.h)

Article 01/31/2022

Which resources are supported for a given format and given device (see [ID3D11Device::CheckFormatSupport](#) and [ID3D11Device::CheckFeatureSupport](#)).

## Syntax

C++

```
typedef enum D3D11_FORMAT_SUPPORT {
    D3D11_FORMAT_SUPPORT_BUFFER = 0x1,
    D3D11_FORMAT_SUPPORT_IA_VERTEX_BUFFER = 0x2,
    D3D11_FORMAT_SUPPORT_IA_INDEX_BUFFER = 0x4,
    D3D11_FORMAT_SUPPORT_SO_BUFFER = 0x8,
    D3D11_FORMAT_SUPPORT_TEXTURE1D = 0x10,
    D3D11_FORMAT_SUPPORT_TEXTURE2D = 0x20,
    D3D11_FORMAT_SUPPORT_TEXTURE3D = 0x40,
    D3D11_FORMAT_SUPPORT_TEXTURECUBE = 0x80,
    D3D11_FORMAT_SUPPORT_SHADER_LOAD = 0x100,
    D3D11_FORMAT_SUPPORT_SHADER_SAMPLE = 0x200,
    D3D11_FORMAT_SUPPORT_SHADER_SAMPLE_COMPARISON = 0x400,
    D3D11_FORMAT_SUPPORT_SHADER_SAMPLE_MONO_TEXT = 0x800,
    D3D11_FORMAT_SUPPORT_MIP = 0x1000,
    D3D11_FORMAT_SUPPORT_MIP_AUTOGEN = 0x2000,
    D3D11_FORMAT_SUPPORT_RENDER_TARGET = 0x4000,
    D3D11_FORMAT_SUPPORT_BLENDABLE = 0x8000,
    D3D11_FORMAT_SUPPORT_DEPTH_STENCIL = 0x10000,
    D3D11_FORMAT_SUPPORT_CPU_LOCKABLE = 0x20000,
    D3D11_FORMAT_SUPPORT_MULTISAMPLE_RESOLVE = 0x40000,
    D3D11_FORMAT_SUPPORT_DISPLAY = 0x80000,
    D3D11_FORMAT_SUPPORT_CAST_WITHIN_BIT_LAYOUT = 0x100000,
    D3D11_FORMAT_SUPPORT_MULTISAMPLE_RENDERTARGET = 0x200000,
    D3D11_FORMAT_SUPPORT_MULTISAMPLE_LOAD = 0x400000,
    D3D11_FORMAT_SUPPORT_SHADER_GATHER = 0x800000,
    D3D11_FORMAT_SUPPORT_BACK_BUFFER_CAST = 0x1000000,
    D3D11_FORMAT_SUPPORT_TYPED_UNORDERED_ACCESS_VIEW = 0x2000000,
    D3D11_FORMAT_SUPPORT_SHADER_GATHER_COMPARISON = 0x4000000,
    D3D11_FORMAT_SUPPORT_DECODER_OUTPUT = 0x8000000,
    D3D11_FORMAT_SUPPORT_VIDEO_PROCESSOR_OUTPUT = 0x10000000,
    D3D11_FORMAT_SUPPORT_VIDEO_PROCESSOR_INPUT = 0x20000000,
    D3D11_FORMAT_SUPPORT_VIDEO_ENCODER = 0x40000000
} ;
```

## Constants

`D3D11_FORMAT_SUPPORT_BUFFER`

Value: *0x1*

Buffer resources supported.

`D3D11_FORMAT_SUPPORT_IA_VERTEX_BUFFER`

Value: *0x2*

Vertex buffers supported.

`D3D11_FORMAT_SUPPORT_IA_INDEX_BUFFER`

Value: *0x4*

Index buffers supported.

`D3D11_FORMAT_SUPPORT_SO_BUFFER`

Value: *0x8*

Streaming output buffers supported.

`D3D11_FORMAT_SUPPORT_TEXTURE1D`

Value: *0x10*

1D texture resources supported.

`D3D11_FORMAT_SUPPORT_TEXTURE2D`

Value: *0x20*

2D texture resources supported.

`D3D11_FORMAT_SUPPORT_TEXTURE3D`

Value: *0x40*

3D texture resources supported.

`D3D11_FORMAT_SUPPORT_TEXTURECUBE`

Value: *0x80*

Cube texture resources supported.

`D3D11_FORMAT_SUPPORT_SHADER_LOAD`

Value: *0x100*

The HLSL [Load](#) function for texture objects is supported.

`D3D11_FORMAT_SUPPORT_SHADER_SAMPLE`

Value: *0x200*

The HLSL [Sample](#) function for texture objects is supported.

**Note** If the device supports the format as a resource (1D, 2D, 3D, or cube map) but doesn't support this option, the resource can still use the **Sample** method but must use only the point filtering sampler state to perform the sample.

`D3D11_FORMAT_SUPPORT_SHADER_SAMPLE_COMPARISON`

Value: *0x400*

The HLSL [SampleCmp](#) and [SampleCmpLevelZero](#) functions for texture objects are supported.

**Note** Windows 8 and later might provide limited support for these functions on Direct3D feature levels 9\_1, 9\_2, and 9\_3. For more info, see [Implementing shadow buffers for Direct3D feature level 9](#).

`D3D11_FORMAT_SUPPORT_SHADER_SAMPLE_MONO_TEXT`

Value: *0x800*

Reserved.

`D3D11_FORMAT_SUPPORT_MIP`

Value: *0x1000*

Mipmaps are supported.

`D3D11_FORMAT_SUPPORT_MIP_AUTOGEN`

Value: *0x2000*

Automatic generation of mipmaps is supported.

`D3D11_FORMAT_SUPPORT_RENDER_TARGET`

Value: *0x4000*

Render targets are supported.

`D3D11_FORMAT_SUPPORT_BLENDABLE`

Value: *0x8000*

Blend operations supported.

`D3D11_FORMAT_SUPPORT_DEPTH_STENCIL`

Value: *0x10000*

Depth stencils supported.

`D3D11_FORMAT_SUPPORT_CPU_LOCKABLE`

Value: *0x20000*

CPU locking supported.

`D3D11_FORMAT_SUPPORT_MULTISAMPLE_RESOLVE`

Value: *0x40000*

Multisample antialiasing (MSAA) resolve operations are supported. For more info, see [ID3D11DeviceContext::ResolveSubresource](#).

`D3D11_FORMAT_SUPPORT_DISPLAY`

Value: `0x80000`

Format can be displayed on screen.

`D3D11_FORMAT_SUPPORT_CAST_WITHIN_BIT_LAYOUT`

Value: `0x100000`

Format cannot be cast to another format.

`D3D11_FORMAT_SUPPORT_MULTISAMPLE_RENDERTARGET`

Value: `0x200000`

Format can be used as a multisampled rendertarget.

`D3D11_FORMAT_SUPPORT_MULTISAMPLE_LOAD`

Value: `0x400000`

Format can be used as a multisampled texture and read into a shader with the HLSL load function.

`D3D11_FORMAT_SUPPORT_SHADER_GATHER`

Value: `0x800000`

Format can be used with the HLSL gather function. This value is available in DirectX 10.1 or higher.

`D3D11_FORMAT_SUPPORT_BACK_BUFFER_CAST`

Value: `0x1000000`

Format supports casting when the resource is a back buffer.

`D3D11_FORMAT_SUPPORT_TYPED_UNORDERED_ACCESS_VIEW`

Value: `0x2000000`

Format can be used for an unordered access view.

`D3D11_FORMAT_SUPPORT_SHADER_GATHER_COMPARISON`

Value: `0x4000000`

Format can be used with the HLSL gather with comparison function.

`D3D11_FORMAT_SUPPORT_DECODER_OUTPUT`

Value: `0x8000000`

Format can be used with the decoder output.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

`D3D11_FORMAT_SUPPORT_VIDEO_PROCESSOR_OUTPUT`

Value: `0x10000000`

Format can be used with the video processor output.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_FORMAT\_SUPPORT\_VIDEO\_PROCESSOR\_INPUT**

Value: 0x20000000

Format can be used with the video processor input.

Direct3D 11: This value is not supported until Direct3D 11.1.

**D3D11\_FORMAT\_SUPPORT\_VIDEO\_ENCODER**

Value: 0x40000000

Format can be used with the video encoder.

Direct3D 11: This value is not supported until Direct3D 11.1.

## Requirements

Header

d3d11.h

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_FORMAT\_SUPPORT2 enumeration (d3d11.h)

Article02/10/2023

Unordered resource support options for a compute shader resource (see [ID3D11Device::CheckFeatureSupport](#)).

## Syntax

C++

```
typedef enum D3D11_FORMAT_SUPPORT2 {
    D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_ADD = 0x1,
    D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_BITWISE_OPS = 0x2,
    D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_COMPARE_STORE_OR_COMPARE_EXCHANGE = 0x4,
    D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_EXCHANGE = 0x8,
    D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_SIGNED_MIN_OR_MAX = 0x10,
    D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_UNSIGNED_MIN_OR_MAX = 0x20,
    D3D11_FORMAT_SUPPORT2_UAV_TYPED_LOAD = 0x40,
    D3D11_FORMAT_SUPPORT2_UAV_TYPED_STORE = 0x80,
    D3D11_FORMAT_SUPPORT2_OUTPUT_MERGER_LOGIC_OP = 0x100,
    D3D11_FORMAT_SUPPORT2_TILED = 0x200,
    D3D11_FORMAT_SUPPORT2_SHAREABLE = 0x400,
    D3D11_FORMAT_SUPPORT2_MULTIPLANE_OVERLAY = 0x4000,
    D3D11_FORMAT_SUPPORT2_DISPLAYABLE
} ;
```

## Constants

`D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_ADD`

Value: `0x1`

Format supports atomic add.

`D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_BITWISE_OPS`

Value: `0x2`

Format supports atomic bitwise operations.

`D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_COMPARE_STORE_OR_COMPARE_EXCHANGE`

Value: `0x4`

Format supports atomic compare with store or exchange.

`D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_EXCHANGE`

Value: *0x8*

Format supports atomic exchange.

`D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_SIGNED_MIN_OR_MAX`

Value: *0x10*

Format supports atomic min and max.

`D3D11_FORMAT_SUPPORT2_UAV_ATOMIC_UNSIGNED_MIN_OR_MAX`

Value: *0x20*

Format supports atomic unsigned min and max.

`D3D11_FORMAT_SUPPORT2_UAV_TYPED_LOAD`

Value: *0x40*

Format supports a typed load.

`D3D11_FORMAT_SUPPORT2_UAV_TYPED_STORE`

Value: *0x80*

Format supports a typed store.

`D3D11_FORMAT_SUPPORT2_OUTPUT_MERGER_LOGIC_OP`

Value: *0x100*

Format supports logic operations in blend state.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

`D3D11_FORMAT_SUPPORT2_TILED`

Value: *0x200*

Format supports tiled resources.

**Direct3D 11:** This value is not supported until Direct3D 11.2.

`D3D11_FORMAT_SUPPORT2_SHAREABLE`

Value: *0x400*

Format supports shareable resources.

Note `DXGI_FORMAT_R8G8B8A8_UNORM` and `DXGI_FORMAT_R8G8B8A8_UNORM_SRGB` are never shareable when using feature level 9, even if the device indicates optional feature support for `D3D11_FORMAT_SUPPORT_SHAREABLE`.

Attempting to create shared resources with DXGI formats `DXGI_FORMAT_R8G8B8A8_UNORM` and `DXGI_FORMAT_R8G8B8A8_UNORM_SRGB` will always fail unless the feature level is 10\_0 or higher.

Direct3D 11: This value is not supported until Direct3D 11.2.

`D3D11_FORMAT_SUPPORT2_MULTIPLANE_OVERLAY`

Value: *0x4000*

Format supports multi-plane overlays.

## Requirements

Header

d3d11.h

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_INPUT\_CLASSIFICATION enumeration (d3d11.h)

Article02/22/2024

Type of data contained in an input slot.

## Syntax

C++

```
typedef enum D3D11_INPUT_CLASSIFICATION {
    D3D11_INPUT_PER_VERTEX_DATA = 0,
    D3D11_INPUT_PER_INSTANCE_DATA = 1
};
```

## Constants

[+] Expand table

<code>D3D11_INPUT_PER_VERTEX_DATA</code>
--

Value: 0

Input data is per-vertex data.

<code>D3D11_INPUT_PER_INSTANCE_DATA</code>
--

Value: 1

Input data is per-instance data.

## Remarks

Use these values to specify the type of data for a particular input element (see [D3D11\\_INPUT\\_ELEMENT\\_DESC](#)) of an input-layout object.

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?



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

# D3D11\_LOGIC\_OP enumeration (d3d11\_1.h)

Article07/27/2022

**Note** This enumeration is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

Specifies logical operations to configure for a render target.

## Syntax

C++

```
typedef enum D3D11_LOGIC_OP {
    D3D11_LOGIC_OP_CLEAR = 0,
    D3D11_LOGIC_OP_SET,
    D3D11_LOGIC_OP_COPY,
    D3D11_LOGIC_OP_COPY_INVERTED,
    D3D11_LOGIC_OP_NOOP,
    D3D11_LOGIC_OP_INVERT,
    D3D11_LOGIC_OP_AND,
    D3D11_LOGIC_OP_NAND,
    D3D11_LOGIC_OP_OR,
    D3D11_LOGIC_OP_NOR,
    D3D11_LOGIC_OP_XOR,
    D3D11_LOGIC_OP_EQUIV,
    D3D11_LOGIC_OP_AND_REVERSE,
    D3D11_LOGIC_OP_AND_INVERTED,
    D3D11_LOGIC_OP_OR_REVERSE,
    D3D11_LOGIC_OP_OR_INVERTED
} ;
```

## Constants

`D3D11_LOGIC_OP_CLEAR`

Value: 0

Clears the render target.

`D3D11_LOGIC_OP_SET`

Sets the render target.

**D3D11\_LOGIC\_OP\_COPY**

Copies the render target.

**D3D11\_LOGIC\_OP\_COPY\_INVERTED**

Performs an inverted-copy of the render target.

**D3D11\_LOGIC\_OP\_NOOP**

No operation is performed on the render target.

**D3D11\_LOGIC\_OP\_INVERT**

Inverts the render target.

**D3D11\_LOGIC\_OP\_AND**

Performs a logical AND operation on the render target.

**D3D11\_LOGIC\_OP\_NAND**

Performs a logical NAND operation on the render target.

**D3D11\_LOGIC\_OP\_OR**

Performs a logical OR operation on the render target.

**D3D11\_LOGIC\_OP\_NOR**

Performs a logical NOR operation on the render target.

**D3D11\_LOGIC\_OP\_XOR**

Performs a logical XOR operation on the render target.

**D3D11\_LOGIC\_OP\_EQUIV**

Performs a logical equal operation on the render target.

**D3D11\_LOGIC\_OP\_AND\_REVERSE**

Performs a logical AND and reverse operation on the render target.

**D3D11\_LOGIC\_OP\_AND\_INVERTED**

Performs a logical AND and invert operation on the render target.

**D3D11\_LOGIC\_OP\_OR\_REVERSE**

Performs a logical OR and reverse operation on the render target.

**D3D11\_LOGIC\_OP\_OR\_INVERTED**

Performs a logical OR and invert operation on the render target.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11_1.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D\_PRIMITIVE enumeration (d3dcommon.h)

Article01/31/2022

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

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_PRIMITIVE {
    D3D_PRIMITIVE_UNDEFINED = 0,
    D3D_PRIMITIVE_POINT = 1,
    D3D_PRIMITIVE_LINE = 2,
    D3D_PRIMITIVE_TRIANGLE = 3,
    D3D_PRIMITIVE_LINE_ADJ = 6,
    D3D_PRIMITIVE_TRIANGLE_ADJ = 7,
    D3D_PRIMITIVE_1_CONTROL_POINT_PATCH = 8,
    D3D_PRIMITIVE_2_CONTROL_POINT_PATCH = 9,
    D3D_PRIMITIVE_3_CONTROL_POINT_PATCH = 10,
    D3D_PRIMITIVE_4_CONTROL_POINT_PATCH = 11,
    D3D_PRIMITIVE_5_CONTROL_POINT_PATCH = 12,
    D3D_PRIMITIVE_6_CONTROL_POINT_PATCH = 13,
    D3D_PRIMITIVE_7_CONTROL_POINT_PATCH = 14,
    D3D_PRIMITIVE_8_CONTROL_POINT_PATCH = 15,
    D3D_PRIMITIVE_9_CONTROL_POINT_PATCH = 16,
    D3D_PRIMITIVE_10_CONTROL_POINT_PATCH = 17,
    D3D_PRIMITIVE_11_CONTROL_POINT_PATCH = 18,
    D3D_PRIMITIVE_12_CONTROL_POINT_PATCH = 19,
    D3D_PRIMITIVE_13_CONTROL_POINT_PATCH = 20,
    D3D_PRIMITIVE_14_CONTROL_POINT_PATCH = 21,
    D3D_PRIMITIVE_15_CONTROL_POINT_PATCH = 22,
    D3D_PRIMITIVE_16_CONTROL_POINT_PATCH = 23,
    D3D_PRIMITIVE_17_CONTROL_POINT_PATCH = 24,
    D3D_PRIMITIVE_18_CONTROL_POINT_PATCH = 25,
    D3D_PRIMITIVE_19_CONTROL_POINT_PATCH = 26,
    D3D_PRIMITIVE_20_CONTROL_POINT_PATCH = 27,
    D3D_PRIMITIVE_21_CONTROL_POINT_PATCH = 28,
    D3D_PRIMITIVE_22_CONTROL_POINT_PATCH = 29,
    D3D_PRIMITIVE_23_CONTROL_POINT_PATCH = 30,
    D3D_PRIMITIVE_24_CONTROL_POINT_PATCH = 31,
```

```
D3D_PRIMITIVE_25_CONTROL_POINT_PATCH = 32,
D3D_PRIMITIVE_26_CONTROL_POINT_PATCH = 33,
D3D_PRIMITIVE_27_CONTROL_POINT_PATCH = 34,
D3D_PRIMITIVE_28_CONTROL_POINT_PATCH = 35,
D3D_PRIMITIVE_29_CONTROL_POINT_PATCH = 36,
D3D_PRIMITIVE_30_CONTROL_POINT_PATCH = 37,
D3D_PRIMITIVE_31_CONTROL_POINT_PATCH = 38,
D3D_PRIMITIVE_32_CONTROL_POINT_PATCH = 39,
D3D10_PRIMITIVE_UNDEFINED,
D3D10_PRIMITIVE_POINT,
D3D10_PRIMITIVE_LINE,
D3D10_PRIMITIVE_TRIANGLE,
D3D10_PRIMITIVE_LINE_ADJ,
D3D10_PRIMITIVE_TRIANGLE_ADJ,
D3D11_PRIMITIVE_UNDEFINED,
D3D11_PRIMITIVE_POINT,
D3D11_PRIMITIVE_LINE,
D3D11_PRIMITIVE_TRIANGLE,
D3D11_PRIMITIVE_LINE_ADJ,
D3D11_PRIMITIVE_TRIANGLE_ADJ,
D3D11_PRIMITIVE_1_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_2_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_3_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_4_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_5_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_6_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_7_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_8_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_9_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_10_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_11_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_12_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_13_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_14_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_15_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_16_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_17_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_18_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_19_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_20_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_21_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_22_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_23_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_24_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_25_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_26_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_27_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_28_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_29_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_30_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_31_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_32_CONTROL_POINT_PATCH
} ;
```

# Constants

[Expand table](#)

D3D_PRIMITIVE_UNDEFINED
Value: 0
D3D_PRIMITIVE_POINT
Value: 1
D3D_PRIMITIVE_LINE
Value: 2
D3D_PRIMITIVE_TRIANGLE
Value: 3
D3D_PRIMITIVE_LINE_ADJ
Value: 6
D3D_PRIMITIVE_TRIANGLE_ADJ
Value: 7
D3D_PRIMITIVE_1_CONTROL_POINT_PATCH
Value: 8
D3D_PRIMITIVE_2_CONTROL_POINT_PATCH
Value: 9
D3D_PRIMITIVE_3_CONTROL_POINT_PATCH
Value: 10
D3D_PRIMITIVE_4_CONTROL_POINT_PATCH
Value: 11
D3D_PRIMITIVE_5_CONTROL_POINT_PATCH
Value: 12
D3D_PRIMITIVE_6_CONTROL_POINT_PATCH
Value: 13
D3D_PRIMITIVE_7_CONTROL_POINT_PATCH
Value: 14
D3D_PRIMITIVE_8_CONTROL_POINT_PATCH
Value: 15
D3D_PRIMITIVE_9_CONTROL_POINT_PATCH
Value: 16

D3D\_PRIMITIVE\_10\_CONTROL\_POINT\_PATCH

Value: 17

D3D\_PRIMITIVE\_11\_CONTROL\_POINT\_PATCH

Value: 18

D3D\_PRIMITIVE\_12\_CONTROL\_POINT\_PATCH

Value: 19

D3D\_PRIMITIVE\_13\_CONTROL\_POINT\_PATCH

Value: 20

D3D\_PRIMITIVE\_14\_CONTROL\_POINT\_PATCH

Value: 21

D3D\_PRIMITIVE\_15\_CONTROL\_POINT\_PATCH

Value: 22

D3D\_PRIMITIVE\_16\_CONTROL\_POINT\_PATCH

Value: 23

D3D\_PRIMITIVE\_17\_CONTROL\_POINT\_PATCH

Value: 24

D3D\_PRIMITIVE\_18\_CONTROL\_POINT\_PATCH

Value: 25

D3D\_PRIMITIVE\_19\_CONTROL\_POINT\_PATCH

Value: 26

D3D\_PRIMITIVE\_20\_CONTROL\_POINT\_PATCH

Value: 27

D3D\_PRIMITIVE\_21\_CONTROL\_POINT\_PATCH

Value: 28

D3D\_PRIMITIVE\_22\_CONTROL\_POINT\_PATCH

Value: 29

D3D\_PRIMITIVE\_23\_CONTROL\_POINT\_PATCH

Value: 30

D3D\_PRIMITIVE\_24\_CONTROL\_POINT\_PATCH

Value: 31

D3D\_PRIMITIVE\_25\_CONTROL\_POINT\_PATCH

Value: 32

D3D\_PRIMITIVE\_26\_CONTROL\_POINT\_PATCH

Value: 33

D3D\_PRIMITIVE\_27\_CONTROL\_POINT\_PATCH

Value: 34

D3D\_PRIMITIVE\_28\_CONTROL\_POINT\_PATCH

Value: 35

D3D\_PRIMITIVE\_29\_CONTROL\_POINT\_PATCH

Value: 36

D3D\_PRIMITIVE\_30\_CONTROL\_POINT\_PATCH

Value: 37

D3D\_PRIMITIVE\_31\_CONTROL\_POINT\_PATCH

Value: 38

D3D\_PRIMITIVE\_32\_CONTROL\_POINT\_PATCH

Value: 39

D3D10\_PRIMITIVE\_UNDEFINED

D3D10\_PRIMITIVE\_POINT

D3D10\_PRIMITIVE\_LINE

D3D10\_PRIMITIVE\_TRIANGLE

D3D10\_PRIMITIVE\_LINE\_ADJ

D3D10\_PRIMITIVE\_TRIANGLE\_ADJ

D3D11\_PRIMITIVE\_UNDEFINED

The shader has not been initialized with an input primitive type.

D3D11\_PRIMITIVE\_POINT

Interpret the input primitive as a point.

D3D11\_PRIMITIVE\_LINE

Interpret the input primitive as a line.

D3D11\_PRIMITIVE\_TRIANGLE

Interpret the input primitive as a triangle.

D3D11\_PRIMITIVE\_LINE\_ADJ

Interpret the input primitive as a line with adjacency data.

D3D11\_PRIMITIVE\_TRIANGLE\_ADJ

Interpret the input primitive as a triangle with adjacency data.

**D3D11\_PRIMITIVE\_1\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_2\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_3\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_4\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_5\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_6\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_7\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_8\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_9\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_10\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_11\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_12\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_13\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_14\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_15\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_16\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_17\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_18\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_19\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_20\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_21\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_22\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_23\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_24\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_25\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_26\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_27\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_28\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_29\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_30\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_31\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_32\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

## Remarks

The [ID3D11ShaderReflection::GetGSInputPrimitive](#) method returns a **D3D11\_PRIMITIVE**-typed value.

The **D3D11\_PRIMITIVE** enumeration is type defined in the D3D11.h header file as a [D3D\\_PRIMITIVE](#) enumeration, which is fully defined in the D3DCommon.h header file.

```
typedef D3D_PRIMITIVE D3D11_PRIMITIVE;
```

## Requirements

 [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Core Enumerations](#)

# D3D11\_PRIMITIVE\_TOPOLOGY enumeration

Article 12/01/2017

How the pipeline interprets vertex data that is bound to the input-assembler stage. These primitive topology values determine how the vertex data is rendered on screen.

## Syntax

C++

```
typedef enum D3D11_PRIMITIVE_TOPOLOGY {
    D3D11_PRIMITIVE_TOPOLOGY_UNDEFINED = 0,
    D3D11_PRIMITIVE_TOPOLOGY_POINTLIST = 1,
    D3D11_PRIMITIVE_TOPOLOGY_LINELIST = 2,
    D3D11_PRIMITIVE_TOPOLOGY_LINESTRIP = 3,
    D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST = 4,
    D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP = 5,
    D3D11_PRIMITIVE_TOPOLOGY_LINELIST_ADJ = 10,
    D3D11_PRIMITIVE_TOPOLOGY_LINESTRIP_ADJ = 11,
    D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST_ADJ = 12,
    D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_ADJ = 13,
    D3D11_PRIMITIVE_TOPOLOGY_1_CONTROL_POINT_PATCHLIST = 33,
    D3D11_PRIMITIVE_TOPOLOGY_2_CONTROL_POINT_PATCHLIST = 34,
    D3D11_PRIMITIVE_TOPOLOGY_3_CONTROL_POINT_PATCHLIST = 35,
    D3D11_PRIMITIVE_TOPOLOGY_4_CONTROL_POINT_PATCHLIST = 36,
    D3D11_PRIMITIVE_TOPOLOGY_5_CONTROL_POINT_PATCHLIST = 37,
    D3D11_PRIMITIVE_TOPOLOGY_6_CONTROL_POINT_PATCHLIST = 38,
    D3D11_PRIMITIVE_TOPOLOGY_7_CONTROL_POINT_PATCHLIST = 39,
    D3D11_PRIMITIVE_TOPOLOGY_8_CONTROL_POINT_PATCHLIST = 40,
    D3D11_PRIMITIVE_TOPOLOGY_9_CONTROL_POINT_PATCHLIST = 41,
    D3D11_PRIMITIVE_TOPOLOGY_10_CONTROL_POINT_PATCHLIST = 42,
    D3D11_PRIMITIVE_TOPOLOGY_11_CONTROL_POINT_PATCHLIST = 43,
    D3D11_PRIMITIVE_TOPOLOGY_12_CONTROL_POINT_PATCHLIST = 44,
    D3D11_PRIMITIVE_TOPOLOGY_13_CONTROL_POINT_PATCHLIST = 45,
    D3D11_PRIMITIVE_TOPOLOGY_14_CONTROL_POINT_PATCHLIST = 46,
    D3D11_PRIMITIVE_TOPOLOGY_15_CONTROL_POINT_PATCHLIST = 47,
    D3D11_PRIMITIVE_TOPOLOGY_16_CONTROL_POINT_PATCHLIST = 48,
    D3D11_PRIMITIVE_TOPOLOGY_17_CONTROL_POINT_PATCHLIST = 49,
    D3D11_PRIMITIVE_TOPOLOGY_18_CONTROL_POINT_PATCHLIST = 50,
    D3D11_PRIMITIVE_TOPOLOGY_19_CONTROL_POINT_PATCHLIST = 51,
    D3D11_PRIMITIVE_TOPOLOGY_20_CONTROL_POINT_PATCHLIST = 52,
    D3D11_PRIMITIVE_TOPOLOGY_21_CONTROL_POINT_PATCHLIST = 53,
    D3D11_PRIMITIVE_TOPOLOGY_22_CONTROL_POINT_PATCHLIST = 54,
    D3D11_PRIMITIVE_TOPOLOGY_23_CONTROL_POINT_PATCHLIST = 55,
    D3D11_PRIMITIVE_TOPOLOGY_24_CONTROL_POINT_PATCHLIST = 56,
    D3D11_PRIMITIVE_TOPOLOGY_25_CONTROL_POINT_PATCHLIST = 57,
    D3D11_PRIMITIVE_TOPOLOGY_26_CONTROL_POINT_PATCHLIST = 58,
    D3D11_PRIMITIVE_TOPOLOGY_27_CONTROL_POINT_PATCHLIST = 59,
    D3D11_PRIMITIVE_TOPOLOGY_28_CONTROL_POINT_PATCHLIST = 60,
    D3D11_PRIMITIVE_TOPOLOGY_29_CONTROL_POINT_PATCHLIST = 61,
    D3D11_PRIMITIVE_TOPOLOGY_30_CONTROL_POINT_PATCHLIST = 62,
```

```
D3D11_PRIMITIVE_TOPOLOGY_31_CONTROL_POINT_PATCHLIST = 63,  
D3D11_PRIMITIVE_TOPOLOGY_32_CONTROL_POINT_PATCHLIST = 64  
} D3D11_PRIMITIVE_TOPOLOGY;
```

## Constants

- **D3D11\_PRIMITIVE\_TOPOLOGY\_UNDEFINED**  
The IA stage has not been initialized with a primitive topology. The IA stage will not function properly unless a primitive topology is defined.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_POINTLIST**  
Interpret the vertex data as a list of points.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_LINELIST**  
Interpret the vertex data as a list of lines.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_LINESTRIP**  
Interpret the vertex data as a line strip.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST**  
Interpret the vertex data as a list of triangles.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLESTRIP**  
Interpret the vertex data as a triangle strip.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_LINELIST\_ADJ**  
Interpret the vertex data as list of lines with adjacency data.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_LINESTRIP\_ADJ**  
Interpret the vertex data as line strip with adjacency data.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST\_ADJ**  
Interpret the vertex data as list of triangles with adjacency data.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLESTRIP\_ADJ**  
Interpret the vertex data as triangle strip with adjacency data.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_1\_CONTROL\_POINT\_PATCHLIST**  
Interpret the vertex data as a patch list.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_2\_CONTROL\_POINT\_PATCHLIST**  
Interpret the vertex data as a patch list.
- **D3D11\_PRIMITIVE\_TOPOLOGY\_3\_CONTROL\_POINT\_PATCHLIST**  
Interpret the vertex data as a patch list.

- D3D11\_PRIMITIVE\_TOPOLOGY\_4\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_5\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_6\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_7\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_8\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_9\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_10\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_11\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_12\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_13\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_14\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_15\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_16\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_17\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_18\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.

- D3D11\_PRIMITIVE\_TOPOLOGY\_19\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_20\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_21\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_22\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_23\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_24\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_25\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_26\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_27\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_28\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_29\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_30\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_31\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.
- D3D11\_PRIMITIVE\_TOPOLOGY\_32\_CONTROL\_POINT\_PATCHLIST  
Interpret the vertex data as a patch list.

## Remarks

The D3D11\_PRIMITIVE\_TOPOLOGY enumeration is type defined in the D3D11.h header file as a [D3D\\_PRIMITIVE\\_TOPOLOGY](#) enumeration, which is fully defined in the D3DCommon.h header file.

```
typedef D3D_PRIMITIVE_TOPOLOGY D3D11_PRIMITIVE_TOPOLOGY;
```

## Requirements

 [Expand table](#)

Header	D3D11.h
--------	---------

## See also

[Core Enumerations](#)

# D3D11\_QUERY enumeration (d3d11.h)

Article 07/27/2022

Query types.

## Syntax

C++

```
typedef enum D3D11_QUERY {
    D3D11_QUERY_EVENT = 0,
    D3D11_QUERY_OCCLUSION,
    D3D11_QUERY_TIMESTAMP,
    D3D11_QUERY_TIMESTAMP_DISJOINT,
    D3D11_QUERY_PIPELINE_STATISTICS,
    D3D11_QUERY_OCCLUSION_PREDICATE,
    D3D11_QUERY_SO_STATISTICS,
    D3D11_QUERY_SO_OVERFLOW_PREDICATE,
    D3D11_QUERY_SO_STATISTICS_STREAM0,
    D3D11_QUERY_SO_OVERFLOW_PREDICATE_STREAM0,
    D3D11_QUERY_SO_STATISTICS_STREAM1,
    D3D11_QUERY_SO_OVERFLOW_PREDICATE_STREAM1,
    D3D11_QUERY_SO_STATISTICS_STREAM2,
    D3D11_QUERY_SO_OVERFLOW_PREDICATE_STREAM2,
    D3D11_QUERY_SO_STATISTICS_STREAM3,
    D3D11_QUERY_SO_OVERFLOW_PREDICATE_STREAM3
} ;
```

## Constants

`D3D11_QUERY_EVENT`

Value: 0

Determines whether or not the GPU is finished processing commands. When the GPU is finished processing commands [ID3D11DeviceContext::GetData](#) will return S\_OK, and pData will point to a BOOL with a value of TRUE. When using this type of query, [ID3D11DeviceContext::Begin](#) is disabled.

`D3D11_QUERY_OCCLUSION`

Get the number of samples that passed the depth and stencil tests in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) returns a UINT64. If a depth or stencil test is disabled, then each of those tests will be counted as a pass.

#### D3D11\_QUERY\_TIMESTAMP

Get a timestamp value where [ID3D11DeviceContext::GetData](#) returns a `UINT64`. This kind of query is only useful if two timestamp queries are done in the middle of a `D3D11_QUERY_TIMESTAMP_DISJOINT` query. The difference of two timestamps can be used to determine how many ticks have elapsed, and the `D3D11_QUERY_TIMESTAMP_DISJOINT` query will determine if that difference is a reliable value and also has a value that shows how to convert the number of ticks into seconds. See [D3D11\\_QUERY\\_DATA\\_TIMESTAMP\\_DISJOINT](#). When using this type of query, [ID3D11DeviceContext::Begin](#) is disabled.

#### D3D11\_QUERY\_TIMESTAMP\_DISJOINT

Determines whether or not a `D3D11_QUERY_TIMESTAMP` is returning reliable values, and also gives the frequency of the processor enabling you to convert the number of elapsed ticks into seconds. [ID3D11DeviceContext::GetData](#) will return a [D3D11\\_QUERY\\_DATA\\_TIMESTAMP\\_DISJOINT](#). This type of query should only be invoked once per frame or less.

#### D3D11\_QUERY\_PIPELINE\_STATISTICS

Get pipeline statistics, such as the number of pixel shader invocations in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) will return a [D3D11\\_QUERY\\_DATA\\_PIPELINE\\_STATISTICS](#).

#### D3D11\_QUERY\_OCCLUSION\_PREDICATE

Similar to `D3D11_QUERY_OCCLUSION`, except [ID3D11DeviceContext::GetData](#) returns a `BOOL` indicating whether or not any samples passed the depth and stencil tests - `TRUE` meaning at least one passed, `FALSE` meaning none passed.

#### D3D11\_QUERY\_SO\_STATISTICS

Get streaming output statistics, such as the number of primitives streamed out in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) will return a [D3D11\\_QUERY\\_DATA\\_SO\\_STATISTICS](#) structure.

#### D3D11\_QUERY\_SO\_OVERFLOW\_PREDICATE

Determines whether or not any of the streaming output buffers overflowed in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) returns a `BOOL` - `TRUE` meaning there was an overflow, `FALSE` meaning there was not an overflow. If streaming output writes to multiple buffers, and one of the buffers overflows, then it will stop writing to all the output buffers. When an overflow is detected by Direct3D it is prevented from happening - no memory is corrupted. This predication may be used in conjunction with an `SO_STATISTICS` query so that when an overflow occurs the `SO_STATISTIC` query will let the application know how much memory was needed to prevent an overflow.

#### D3D11\_QUERY\_SO\_STATISTICS\_STREAM0

Get streaming output statistics for stream 0, such as the number of primitives streamed out in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) will return a [D3D11\\_QUERY\\_DATA\\_SO\\_STATISTICS](#) structure.

#### `D3D11_QUERY_SO_OVERFLOW_PREDICATE_STREAM0`

Determines whether or not the stream 0 output buffers overflowed in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) returns a BOOL - **TRUE** meaning there was an overflow, **FALSE** meaning there was not an overflow. If streaming output writes to multiple buffers, and one of the buffers overflows, then it will stop writing to all the output buffers. When an overflow is detected by Direct3D it is prevented from happening - no memory is corrupted. This predication may be used in conjunction with an SO\_STATISTICS query so that when an overflow occurs the SO\_STATISTIC query will let the application know how much memory was needed to prevent an overflow.

#### `D3D11_QUERY_SO_STATISTICS_STREAM1`

Get streaming output statistics for stream 1, such as the number of primitives streamed out in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#).

[ID3D11DeviceContext::GetData](#) will return a [D3D11\\_QUERY\\_DATA\\_SO\\_STATISTICS](#) structure.

#### `D3D11_QUERY_SO_OVERFLOW_PREDICATE_STREAM1`

Determines whether or not the stream 1 output buffers overflowed in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) returns a BOOL - **TRUE** meaning there was an overflow, **FALSE** meaning there was not an overflow. If streaming output writes to multiple buffers, and one of the buffers overflows, then it will stop writing to all the output buffers. When an overflow is detected by Direct3D it is prevented from happening - no memory is corrupted. This predication may be used in conjunction with an SO\_STATISTICS query so that when an overflow occurs the SO\_STATISTIC query will let the application know how much memory was needed to prevent an overflow.

#### `D3D11_QUERY_SO_STATISTICS_STREAM2`

Get streaming output statistics for stream 2, such as the number of primitives streamed out in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#).

[ID3D11DeviceContext::GetData](#) will return a [D3D11\\_QUERY\\_DATA\\_SO\\_STATISTICS](#) structure.

#### `D3D11_QUERY_SO_OVERFLOW_PREDICATE_STREAM2`

Determines whether or not the stream 2 output buffers overflowed in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) returns a BOOL - **TRUE** meaning there was an overflow, **FALSE** meaning there was not an overflow. If streaming output writes to multiple buffers, and one of the buffers overflows, then it will stop writing to all the output buffers. When an overflow is detected by Direct3D it is prevented from happening - no memory is corrupted. This predication may be used in conjunction with an SO\_STATISTICS query so that when an overflow occurs the SO\_STATISTIC query will let the application know how much memory was needed to prevent an overflow.

#### `D3D11_QUERY_SO_STATISTICS_STREAM3`

Get streaming output statistics for stream 3, such as the number of primitives streamed out in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#).

[ID3D11DeviceContext::GetData](#) will return a [D3D11\\_QUERY\\_DATA\\_SO\\_STATISTICS](#) structure.

### D3D11\_QUERY\_SO\_OVERFLOW\_PREDICATE\_STREAM3

Determines whether or not the stream 3 output buffers overflowed in between [ID3D11DeviceContext::Begin](#) and [ID3D11DeviceContext::End](#). [ID3D11DeviceContext::GetData](#) returns a BOOL - **TRUE** meaning there was an overflow, **FALSE** meaning there was not an overflow. If streaming output writes to multiple buffers, and one of the buffers overflows, then it will stop writing to all the output buffers. When an overflow is detected by Direct3D it is prevented from happening - no memory is corrupted. This predication may be used in conjunction with an SO\_STATISTICS query so that when an overflow occurs the SO\_STATISTIC query will let the application know how much memory was needed to prevent an overflow.

## Remarks

Create a query with [ID3D11Device::CreateQuery](#).

## Requirements

Header

d3d11.h

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_QUERY\_MISC\_FLAG enumeration (d3d11.h)

Article 02/22/2024

Flags that describe miscellaneous query behavior.

## Syntax

C++

```
typedef enum D3D11_QUERY_MISC_FLAG {
    D3D11_QUERY_MISC_PREDICATEHINT = 0x1
} ;
```

## Constants

[+] Expand table

D3D11\_QUERY\_MISC\_PREDICATEHINT

Value: 0x1

Tell the hardware that if it is not yet sure if something is hidden or not to draw it anyway. This is only used with an occlusion predicate. Predication data cannot be returned to your application via [ID3D11DeviceContext::GetData](#) when using this flag.

## Remarks

This flag is part of a query description (see [D3D11\\_QUERY\\_DESC](#)).

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_RAISE\_FLAG enumeration (d3d11.h)

Article 02/22/2024

Option(s) for raising an error to a non-continuable exception.

## Syntax

C++

```
typedef enum D3D11_RAISE_FLAG {  
    D3D11_RAISE_FLAG_DRIVER_INTERNAL_ERROR = 0x1L  
} ;
```

## Constants

[+] Expand table

D3D11\_RAISE\_FLAG\_DRIVER\_INTERNAL\_ERROR

Value: *0x1L*

Raise an internal driver error to a non-continuable exception.

## Remarks

These flags are used by [ID3D11Device::GetExceptionMode](#) and [ID3D11Device::SetExceptionMode](#). Use 0 to indicate no flags; multiple flags can be logically OR'ed together.

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_SHADER\_CACHE\_SUPPORT\_FLAG

## S enumeration (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef enum D3D11_SHADER_CACHE_SUPPORT_FLAGS {
    D3D11_SHADER_CACHE_SUPPORT_NONE = 0,
    D3D11_SHADER_CACHE_SUPPORT_AUTOMATIC_INPROC_CACHE = 0x1,
    D3D11_SHADER_CACHE_SUPPORT_AUTOMATIC_DISK_CACHE = 0x2
};
```

## Constants

[+] Expand table

D3D11\_SHADER\_CACHE\_SUPPORT\_NONE

Value: 0

Indicates that the driver does not support shader caching.

D3D11\_SHADER\_CACHE\_SUPPORT\_AUTOMATIC\_INPROC\_CACHE

Value: 0x1

Indicates that the driver supports an OS-managed shader cache that stores compiled shaders in memory during the current run of the application.

D3D11\_SHADER\_CACHE\_SUPPORT\_AUTOMATIC\_DISK\_CACHE

Value: 0x2

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 [D3D11\\_FEATURE\\_DATA\\_SHADER\\_CACHE](#) structure.

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SHADER\_MIN\_PRECISION\_SUPP ORT enumeration (d3d11.h)

Article02/22/2024

**Note** This enumeration is supported by the Direct3D 11.1 runtime, which is available on Windows 8 and later operating systems.

Values that specify minimum precision levels at shader stages.

## Syntax

C++

```
typedef enum D3D11_SHADER_MIN_PRECISION_SUPPORT {
    D3D11_SHADER_MIN_PRECISION_10_BIT = 0x1,
    D3D11_SHADER_MIN_PRECISION_16_BIT = 0x2
};
```

## Constants

[+] Expand table

D3D11_SHADER_MIN_PRECISION_10_BIT
-----------------------------------

Value: *0x1*

Minimum precision level is 10-bit.

D3D11_SHADER_MIN_PRECISION_16_BIT
-----------------------------------

Value: *0x2*

Minimum precision level is 16-bit.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

[Core Enumerations](#)

[D3D11\\_FEATURE](#)

[D3D11\\_FEATURE\\_DATA\\_SHADER\\_MIN\\_PRECISION\\_SUPPORT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SHARED\_RESOURCE\_TIER enumeration (d3d11.h)

Article 07/28/2022

Defines constants that specify the level of support for shared resources in the current graphics driver.

## Syntax

C++

```
typedef enum D3D11_SHARED_RESOURCE_TIER {
    D3D11_SHARED_RESOURCE_TIER_0 = 0,
    D3D11_SHARED_RESOURCE_TIER_1,
    D3D11_SHARED_RESOURCE_TIER_2,
    D3D11_SHARED_RESOURCE_TIER_3
};
```

## Constants

`D3D11_SHARED_RESOURCE_TIER_0`

Value: 0

Specifies the support available when

`D3D11_FEATURE_DATA_D3D11_OPTIONS::ExtendedResourceSharing` is `FALSE` (only very old drivers have this value set to `FALSE`).

See [Extended support for shared Texture2D resources](#).

`D3D11_SHARED_RESOURCE_TIER_1`

Specifies the support available when

`D3D11_FEATURE_DATA_D3D11_OPTIONS::ExtendedResourceSharing` and

`D3D11_FEATURE_DATA_D3D11_OPTIONS4::ExtendedNV12SharedTextureSupported` are `TRUE`.

You can share additional formats; see [Extended support for shared Texture2D resources](#).

Only formats that are still shareable when

`D3D11_FEATURE_DATA_D3D11_OPTIONS::ExtendedResourceSharing == FALSE` can be shared across APIs between Direct3D 11 and Direct3D 12.

Resource formats added by `D3D11_FEATURE_DATA_D3D11_OPTIONS::ExtendedResourceSharing == TRUE` can't be shared across APIs.

#### D3D11\_SHARED\_RESOURCE\_TIER\_2

Specifies the support available when

`D3D11_FEATURE_DATA_D3D11_OPTIONS4::ExtendedNV12SharedTextureSupported` is `TRUE`. Also see [Extended NV12 texture support](#).

See [Extended support for shared Texture2D resources](#).

Sharing across APIs between Direct3D 11 and Direct3D 12 is possible for the

`D3D11_FEATURE_DATA_D3D11_OPTIONS::ExtendedResourceSharing == TRUE` format list.

#### D3D11\_SHARED\_RESOURCE\_TIER\_3

Specifies that `DXGI_FORMAT_R11G11B10_FLOAT` supports NT handle sharing. Also see [CreateSharedHandle](#).

Sharing across APIs between Direct3D 11 and Direct3D 12 is possible for the

`DXGI_FORMAT_R11G11B10_FLOAT` format.

## Requirements

Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d11.h

## See also

- [Extended support for shared Texture2D resources](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_STENCIL\_OP enumeration (d3d11.h)

Article 02/22/2024

The stencil operations that can be performed during depth-stencil testing.

## Syntax

C++

```
typedef enum D3D11_STENCIL_OP {
    D3D11_STENCIL_OP_KEEP = 1,
    D3D11_STENCIL_OP_ZERO = 2,
    D3D11_STENCIL_OP_REPLACE = 3,
    D3D11_STENCIL_OP_INCR_SAT = 4,
    D3D11_STENCIL_OP_DECR_SAT = 5,
    D3D11_STENCIL_OP_INVERT = 6,
    D3D11_STENCIL_OP_INCR = 7,
    D3D11_STENCIL_OP_DECR = 8
} ;
```

## Constants

[+] Expand table

D3D11\_STENCIL\_OP\_KEEP

Value: 1

Keep the existing stencil data.

D3D11\_STENCIL\_OP\_ZERO

Value: 2

Set the stencil data to 0.

D3D11\_STENCIL\_OP\_REPLACE

Value: 3

Set the stencil data to the reference value set by calling  
[ID3D11DeviceContext::OMSetDepthStencilState](#).

D3D11\_STENCIL\_OP\_INCR\_SAT

Value: 4

Increment the stencil value by 1, and clamp the result.

`D3D11_STENCIL_OP_DECR_SAT`

Value: 5

Decrement the stencil value by 1, and clamp the result.

`D3D11_STENCIL_OP_INVERT`

Value: 6

Invert the stencil data.

`D3D11_STENCIL_OP_INCR`

Value: 7

Increment the stencil value by 1, and wrap the result if necessary.

`D3D11_STENCIL_OP_DECR`

Value: 8

Decrement the stencil value by 1, and wrap the result if necessary.

## Requirements

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

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEXTURE\_ADDRESS\_MODE enumeration (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef enum D3D11_TEXTURE_ADDRESS_MODE {
    D3D11_TEXTURE_ADDRESS_WRAP = 1,
    D3D11_TEXTURE_ADDRESS_MIRROR = 2,
    D3D11_TEXTURE_ADDRESS_CLAMP = 3,
    D3D11_TEXTURE_ADDRESS_BORDER = 4,
    D3D11_TEXTURE_ADDRESS_MIRROR_ONCE = 5
};
```

## Constants

[+] Expand table

D3D11\_TEXTURE\_ADDRESS\_WRAP

Value: 1

Tile the texture at every (u,v) integer junction. For example, for u values between 0 and 3, the texture is repeated three times.

D3D11\_TEXTURE\_ADDRESS\_MIRROR

Value: 2

Flip the texture at every (u,v) integer junction. For u values between 0 and 1, for example, the texture is addressed normally; between 1 and 2, the texture is flipped (mirrored); between 2 and 3, the texture is normal again; and so on.

D3D11\_TEXTURE\_ADDRESS\_CLAMP

Value: 3

Texture coordinates outside the range [0.0, 1.0] are set to the texture color at 0.0 or 1.0, respectively.

D3D11\_TEXTURE\_ADDRESS\_BORDER

Value: 4

Texture coordinates outside the range [0.0, 1.0] are set to the border color specified in [D3D11\\_SAMPLER\\_DESC](#) or HLSL code.

`D3D11_TEXTURE_ADDRESS_MIRROR_ONCE`

Value: 5

Similar to `D3D11_TEXTURE_ADDRESS_MIRROR` and `D3D11_TEXTURE_ADDRESS_CLAMP`. Takes the absolute value of the texture coordinate (thus, mirroring around 0), and then clamps to the maximum value.

## Requirements

[Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEXTURECUBE\_FACE enumeration (d3d11.h)

Article 02/22/2024

The different faces of a cube texture.

## Syntax

C++

```
typedef enum D3D11_TEXTURECUBE_FACE {
    D3D11_TEXTURECUBE_FACE_POSITIVE_X = 0,
    D3D11_TEXTURECUBE_FACE_NEGATIVE_X = 1,
    D3D11_TEXTURECUBE_FACE_POSITIVE_Y = 2,
    D3D11_TEXTURECUBE_FACE_NEGATIVE_Y = 3,
    D3D11_TEXTURECUBE_FACE_POSITIVE_Z = 4,
    D3D11_TEXTURECUBE_FACE_NEGATIVE_Z = 5
} ;
```

## Constants

[+] Expand table

D3D11\_TEXTURECUBE\_FACE\_POSITIVE\_X

Value: 0

Positive X face.

D3D11\_TEXTURECUBE\_FACE\_NEGATIVE\_X

Value: 1

Negative X face.

D3D11\_TEXTURECUBE\_FACE\_POSITIVE\_Y

Value: 2

Positive Y face.

D3D11\_TEXTURECUBE\_FACE\_NEGATIVE\_Y

Value: 3

Negative Y face.

D3D11\_TEXTURECUBE\_FACE\_POSITIVE\_Z

Value: 4

Positive Z face.

D3D11\_TEXTURECUBE\_FACE\_NEGATIVE\_Z

Value: 5

Negative Z face.

## Requirements

[Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TILED\_RESOURCES\_TIER enumeration (d3d11.h)

Article02/22/2024

Indicates the tier level at which tiled resources are supported.

## Syntax

C++

```
typedef enum D3D11_TILED_RESOURCES_TIER {
    D3D11_TILED_RESOURCES_NOT_SUPPORTED = 0,
    D3D11_TILED_RESOURCES_TIER_1 = 1,
    D3D11_TILED_RESOURCES_TIER_2 = 2,
    D3D11_TILED_RESOURCES_TIER_3 = 3
};
```

## Constants

[+] Expand table

`D3D11_TILED_RESOURCES_NOT_SUPPORTED`

Value: 0

Tiled resources are not supported.

`D3D11_TILED_RESOURCES_TIER_1`

Value: 1

Tier\_1 tiled resources are supported.

The device supports calls to [CreateTexture2D](#) and so on with the [D3D11\\_RESOURCE\\_MISC\\_TILED](#) flag.

The device supports calls to [CreateBuffer](#) with the [D3D11\\_RESOURCE\\_MISC\\_TILE\\_POOL](#) flag.

If you access tiles (read or write) that are **NULL**-mapped, you get undefined behavior, which includes device-removed. Apps can map all tiles to a single "default" tile to avoid this condition.

`D3D11_TILED_RESOURCES_TIER_2`

Value: 2

Tier\_2 tiled resources are supported.

Superset of Tier\_1 functionality, which includes this additional support:

- On Tier\_1, if 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. On Tier\_2, this guarantee is expanded to include mipmap levels whose size is at least one standard tile shape. For more info, see [D3D11\\_PACKED\\_MIP\\_DESC](#).
- 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\(S,float,int,float,uint\)](#).
- Reading from **NUL**L-mapped tiles treat that sampled value as zero. Writes to **NUL**-mapped tiles are discarded.

#### D3D11\_TILED\_RESOURCES\_TIER\_3

Value: 3

Tier\_3 tiled resources are supported.

Superset of Tier\_2 functionality, Tier 3 is essentially Tier 2 but with the additional support of Texture3D for Tiled Resources.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

[D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS1](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# Layer Reference

Article • 11/04/2020

The Direct3D API defines several layer API elements.

## In this section

Topic	Description
<a href="#">Layer Interfaces</a>	This section contains information about the layer interfaces.
<a href="#">Layer Structures</a>	This section contains information about the layer structures.
<a href="#">Layer Enumerations</a>	This section contains information about the layer enumerations.

## Related topics

[Direct3D 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Layer Interfaces

Article • 08/19/2021

This section contains information about the layer interfaces.

## In this section

Topic	Description
<a href="#">ID3D11Debug</a>	A debug interface controls debug settings, validates pipeline state and can only be used if the debug layer is turned on.
<a href="#">ID3D11InfoQueue</a>	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.
<a href="#">ID3D11RefDefaultTrackingOptions</a>	The default tracking interface sets reference default tracking options.
<a href="#">ID3D11RefTrackingOptions</a>	The tracking interface sets reference tracking options.
<a href="#">ID3D11SwitchToRef</a>	<p>[!Note]</p> <p>The <a href="#">ID3D11SwitchToRef</a> interface and its methods are not supported in Direct3D 11.</p>
<a href="#">ID3D11TracingDevice</a>	The tracing device interface sets shader tracking information, which enables accurate logging and playback of shader execution.

## Related topics

[Layer Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug interface (d3d11sdklayers.h)

Article 07/27/2022

A debug interface controls debug settings, validates pipeline state and can only be used if the debug layer is turned on.

## Inheritance

The **ID3D11Debug** interface inherits from the [IUnknown](#) interface. **ID3D11Debug** also has these types of members:

## Methods

The **ID3D11Debug** interface has these methods.

<a href="#">ID3D11Debug::GetFeatureMask</a>
Get a bitfield of flags that indicates which debug features are on or off. (ID3D11Debug.GetFeatureMask)
<a href="#">ID3D11Debug::GetPresentPerRenderOpDelay</a>
Get the number of milliseconds to sleep after IDXGISwapChain::Present is called.
<a href="#">ID3D11Debug::GetSwapChain</a>
Get the swap chain that the runtime will use for automatically calling IDXGISwapChain::Present.
<a href="#">ID3D11Debug::ReportLiveDeviceObjects</a>
Report information about a device object's lifetime.
<a href="#">ID3D11Debug::SetFeatureMask</a>
Set a bit field of flags that will turn debug features on and off. (ID3D11Debug.SetFeatureMask)
<a href="#">ID3D11Debug::SetPresentPerRenderOpDelay</a>
Set the number of milliseconds to sleep after IDXGISwapChain::Present is called.

### [ID3D11Debug::SetSwapChain](#)

Sets a swap chain that the runtime will use for automatically calling IDXGISwapChain::Present.

### [ID3D11Debug::ValidateContext](#)

Check to see if the draw pipeline state is valid.

### [ID3D11Debug::ValidateContextForDispatch](#)

Verifies whether the dispatch pipeline state is valid.

## Remarks

This interface is obtained by querying it from the [ID3D11Device](#) using [IUnknown::QueryInterface](#).

For more information about the debug layer, see [Debug Layer](#).

**Windows Phone 8:** This API is supported.

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11sdklayers.h

## See also

[IUnknown](#)

[Layer Interfaces](#)

## Feedback

Was this page helpful?  

Get help at Microsoft Q&A

# ID3D11Debug::GetFeatureMask method (d3d11sdklayers.h)

Article 02/22/2024

Get a bitfield of flags that indicates which debug features are on or off.

## Syntax

C++

```
UINT GetFeatureMask();
```

## Return value

Type: [UINT](#)

Mask of feature-mask flags bitwise ORed together. If a flag is present, then that feature will be set to on, otherwise the feature will be set to off. See [ID3D11Debug::SetFeatureMask](#) for a list of possible feature-mask flags.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug::GetPresentPerRenderOpDelay method (d3d11sdklayers.h)

Article 02/22/2024

Get the number of milliseconds to sleep after [IDXGISwapChain::Present](#) is called.

## Syntax

C++

```
UINT GetPresentPerRenderOpDelay();
```

## Return value

Type: [UINT](#)

Number of milliseconds to sleep after Present is called.

## Remarks

Value is set with [ID3D11Debug::SetPresentPerRenderOpDelay](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug Interface](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug::GetSwapChain method (d3d11sdklayers.h)

Article 02/22/2024

Get the swap chain that the runtime will use for automatically calling [IDXGISwapChain::Present](#).

## Syntax

C++

```
HRESULT GetSwapChain(  
    [out] IDXGISwapChain **ppSwapChain  
>;
```

## Parameters

[out] ppSwapChain

Type: [IDXGISwapChain\\*\\*](#)

Swap chain that the runtime will use for automatically calling [IDXGISwapChain::Present](#).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

The swap chain retrieved by this method will only be used if D3D11\_DEBUG\_FEATURE\_PRESENT\_PER\_RENDER\_OP is set in the [feature mask](#).

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug::ReportLiveDeviceObjects method (d3d11sdklayers.h)

Article 02/22/2024

Report information about a device object's lifetime.

## Syntax

C++

```
HRESULT ReportLiveDeviceObjects(  
    D3D11_RLDO_FLAGS Flags  
);
```

## Parameters

Flags

Type: [D3D11\\_RLDO\\_FLAGS](#)

A value from the [D3D11\\_RLDO\\_FLAGS](#) enumeration.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

`ReportLiveDeviceObjects` uses the value in *Flags* to determine the amount of information to report about a device object's lifetime.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug::SetFeatureMask method (d3d11sdklayers.h)

Article 07/27/2022

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

## Syntax

C++

```
HRESULT SetFeatureMask(  
    UINT Mask  
)
```

## Parameters

Mask

Type: [UINT](#)

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. For descriptions of the feature-mask flags, see Remarks.

## Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 11 Return Codes](#).

## Remarks

**Note** If you call this API in a Session 0 process, it returns [DXGI\\_ERROR\\_NOT\\_CURRENTLY\\_AVAILABLE](#).

Setting one of the following feature-mask flags will cause a rendering-operation method (listed below) to do some extra task when called.

D3D11_DEBUG_FEATURE_FINISH_PER_RENDER_OP (0x2)	Application will wait for the GPU to finish processing the rendering operation before continuing.
D3D11_DEBUG_FEATURE_FLUSH_PER_RENDER_OP (0x1)	Runtime will additionally call <a href="#">ID3D11DeviceContext::Flush</a> .
D3D11_DEBUG_FEATURE_PRESENT_PER_RENDER_OP (0x4)	Runtime will call <a href="#">IDXGISwapChain::Present</a> . Presentation of render buffers will occur according to the settings established by prior calls to <a href="#">ID3D11Debug::SetSwapChain</a> and <a href="#">ID3D11Debug::SetPresentPerRenderOpDelay</a> .

These feature-mask flags apply to the following rendering-operation methods:

- [ID3D11DeviceContext::Draw](#)
- [ID3D11DeviceContext::DrawIndexed](#)
- [ID3D11DeviceContext::DrawInstanced](#)
- [ID3D11DeviceContext::DrawIndexedInstanced](#)
- [ID3D11DeviceContext::DrawAuto](#)
- [ID3D11DeviceContext::ClearRenderTargetView](#)
- [ID3D11DeviceContext::ClearDepthStencilView](#)
- [ID3D11DeviceContext::CopySubresourceRegion](#)
- [ID3D11DeviceContext::CopyResource](#)
- [ID3D11DeviceContext::UpdateSubresource](#)
- [ID3D11DeviceContext::GenerateMips](#)
- [ID3D11DeviceContext::ResolveSubresource](#)

By setting one of the following feature-mask flags, you can control the behavior of the [IDXGIDevice2::OfferResources](#) and [IDXGIDevice2::ReclaimResources](#) methods to aid in testing and debugging.

**Note** These flags are supported by the Direct3D 11.1 runtime, which is available starting with Windows 8.

D3D11_DEBUG_FEATURE_ALWAYS_DISCARD_OFFERED_RESOURCE (0x8)	When you call <a href="#">IDXGIDevice2::OfferResources</a> to offer resources while this flag is enabled, their content is always discarded. Use this flag to test code paths that regenerate resource content on reclaim. You cannot use this flag in combination with D3D11_DEBUG_FEATURE_NEVER_DISCARD_OFFERED_RESOURCE.
D3D11_DEBUG_FEATURE_NEVER_DISCARD_OFFERED_RESOURCE (0x10)	When you call <a href="#">IDXGIDevice2::OfferResources</a> to offer resources while this flag is enabled, their content is never discarded. Use this flag to test code paths that do not need to regenerate resource content on reclaim. You cannot use this flag in combination with D3D11_DEBUG_FEATURE_ALWAYS_DISCARD_OFFERED_RESOURCE.

The behavior of the [IDXGIDevice2::OfferResources](#) and [IDXGIDevice2::ReclaimResources](#) methods depends on system-wide memory pressure. Therefore, the scenario where content is lost and must be regenerated is uncommon for most applications. The preceding new options in the Direct3D debug layer let you simulate that scenario consistently and test code paths.

The following flag is supported by the Direct3D 11.1 runtime.

D3D11_DEBUG_FEATURE_AVOID_BEHAVIOR_CHANGING_DEBUG_AIDS (0x40)	Disables the following default debugging behavior.
--	--

When the debug layer is enabled, it performs certain actions to reveal application problems. By setting the D3D11\_DEBUG\_FEATURE\_AVOID\_BEHAVIOR\_CHANGING\_DEBUG\_AIDS feature-mask flag, you can enable the debug layer without getting the following default debugging behavior:

- If an application calls [ID3D11DeviceContext1::DiscardView](#), the runtime fills in the resource with a random color.
- If an application calls [IDXGISwapChain1::Present1](#) with partial presentation parameters, the runtime ignores the partial presentation information.

The following flag is supported by the Direct3D 11.2 runtime.

D3D11_DEBUG_FEATURE_DISABLE_TILED_RESOURCE_MAPPING_TRACKING_AND_VALIDATION (0x80)	Disables the following default debugging behavior.
--	--

By default (that is, without

D3D11\_DEBUG\_FEATURE\_DISABLE\_TILED\_RESOURCE\_MAPPING\_TRACKING\_AND\_VALIDATION set), the debug layer validates the proper usage of all tile mappings for [tiled resources](#) for bound resources for every operation performed on the device context (for example, draw, copy, and so on). Depending on the size of the tiled resources used (if any), this validation can be processor intensive and slow. Apps might want to initially run with tiled resource tile mapping validation on; then, when they determine that the calling pattern is safe, they can disable the validation by setting

D3D11\_DEBUG\_FEATURE\_DISABLE\_TILED\_RESOURCE\_MAPPING\_TRACKING\_AND\_VALIDATION.

If D3D11\_DEBUG\_FEATURE\_DISABLE\_TILED\_RESOURCE\_MAPPING\_TRACKING\_AND\_VALIDATION is set when a tiled resource is created, the debug layer never performs the tracking of tile mapping for that resource for its entire lifetime. Alternatively, if

D3D11\_DEBUG\_FEATURE\_DISABLE\_TILED\_RESOURCE\_MAPPING\_TRACKING\_AND\_VALIDATION is set for any given device context method call (like draw or copy calls) involving tiled resources, the debug layer skips all tile mapping validation for the call.

## Requirements

Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug::SetPresentPerRenderOpDelay method (d3d11sdklayers.h)

Article 02/22/2024

Set the number of milliseconds to sleep after [IDXGISwapChain::Present](#) is called.

## Syntax

C++

```
HRESULT SetPresentPerRenderOpDelay(  
    UINT Milliseconds  
>;
```

## Parameters

Milliseconds

Type: [UINT](#)

Number of milliseconds to sleep after Present is called.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

**Note** If you call this API in a Session 0 process, it returns [DXGI\\_ERROR\\_NOT\\_CURRENTLY\\_AVAILABLE](#).

The application will only sleep if D3D11\_DEBUG\_FEATURE\_PRESENT\_PER\_RENDER\_OP is a set in the [feature mask](#). If that flag is not set the number of milliseconds is set but ignored and the application does not sleep. 10ms is used as a default value if this method is never called.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug::SetSwapChain method (d3d11sdklayers.h)

Article 02/22/2024

Sets a swap chain that the runtime will use for automatically calling [IDXGISwapChain::Present](#).

## Syntax

C++

```
HRESULT SetSwapChain(  
    [in, optional] IDXGISwapChain *pSwapChain  
) ;
```

## Parameters

[in, optional] pSwapChain

Type: [IDXGISwapChain\\*](#)

Swap chain that the runtime will use for automatically calling [IDXGISwapChain::Present](#); must have been created with the DXGI\_SWAP\_EFFECT\_SEQUENTIAL swap-effect flag.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

**Note** If you call this API in a Session 0 process, it returns [DXGI\\_ERROR\\_NOT\\_CURRENTLY\\_AVAILABLE](#).

The swap chain set by this method will only be used if D3D11\_DEBUG\_FEATURE\_PRESENT\_PER\_RENDER\_OP is set in the [feature mask](#).

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug::ValidateContext method (d3d11sdklayers.h)

Article 02/22/2024

Check to see if the draw pipeline state is valid.

## Syntax

C++

```
HRESULT ValidateContext(  
    [in] ID3D11DeviceContext *pContext  
);
```

## Parameters

[in] pContext

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#), that represents a device context.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Use validate prior to calling a draw method (for example, [ID3D11DeviceContext::Draw](#)); validation requires the debug layer.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Debug::ValidateContextForDispatch method (d3d11sdklayers.h)

Article 02/22/2024

Verifies whether the dispatch pipeline state is valid.

## Syntax

C++

```
HRESULT ValidateContextForDispatch(
    [in] ID3D11DeviceContext *pContext
);
```

## Parameters

[in] pContext

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#) that represents a device context.

## Return value

Type: [HRESULT](#)

This method returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

Use this method before you call a dispatch method (for example, [ID3D11DeviceContext::Dispatch](#)). Validation requires the debug layer.

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11Debug](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue interface (d3d11sdklayers.h)

Article 07/27/2022

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

## Inheritance

The **ID3D11InfoQueue** interface inherits from the [IUnknown](#) interface. **ID3D11InfoQueue** also has these types of members:

## Methods

The **ID3D11InfoQueue** interface has these methods.

### [ID3D11InfoQueue::AddApplicationMessage](#)

Add a user-defined message to the message queue and send that message to debug output.  
(ID3D11InfoQueue.AddApplicationMessage)

### [ID3D11InfoQueue::AddMessage](#)

Add a debug message to the message queue and send that message to debug output.

### [ID3D11InfoQueue::AddRetrievalFilterEntries](#)

Add storage filters to the top of the retrieval-filter stack.  
(ID3D11InfoQueue.AddRetrievalFilterEntries)

### [ID3D11InfoQueue::AddStorageFilterEntries](#)

Add storage filters to the top of the storage-filter stack.  
(ID3D11InfoQueue.AddStorageFilterEntries)

### [ID3D11InfoQueue::ClearRetrievalFilter](#)

Remove a retrieval filter from the top of the retrieval-filter stack.  
(ID3D11InfoQueue.ClearRetrievalFilter)

## [ID3D11InfoQueue::ClearStorageFilter](#)

Remove a storage filter from the top of the storage-filter stack.  
(ID3D11InfoQueue.ClearStorageFilter)

## [ID3D11InfoQueue::ClearStoredMessages](#)

Clear all messages from the message queue. (ID3D11InfoQueue.ClearStoredMessages)

## [ID3D11InfoQueue::GetBreakOnCategory](#)

Get a message category to break on when a message with that category passes through the storage filter. (ID3D11InfoQueue.GetBreakOnCategory)

## [ID3D11InfoQueue::GetBreakOnID](#)

Get a message identifier to break on when a message with that identifier passes through the storage filter. (ID3D11InfoQueue.GetBreakOnID)

## [ID3D11InfoQueue::GetBreakOnSeverity](#)

Get a message severity level to break on when a message with that severity level passes through the storage filter. (ID3D11InfoQueue.GetBreakOnSeverity)

## [ID3D11InfoQueue::GetMessage](#)

Get a message from the message queue. (ID3D11InfoQueue.GetMessage)

## [ID3D11InfoQueue::GetMessageCountLimit](#)

Get the maximum number of messages that can be added to the message queue.  
(ID3D11InfoQueue.GetMessageCountLimit)

## [ID3D11InfoQueue::GetMuteDebugOutput](#)

Get a boolean that turns the debug output on or off. (ID3D11InfoQueue.GetMuteDebugOutput)

## [ID3D11InfoQueue::GetNumMessagesAllowedByStorageFilter](#)

Get the number of messages that were allowed to pass through a storage filter.  
(ID3D11InfoQueue.GetNumMessagesAllowedByStorageFilter)

## [ID3D11InfoQueue::GetNumMessagesDeniedByStorageFilter](#)

Get the number of messages that were denied passage through a storage filter.  
(ID3D11InfoQueue.GetNumMessagesDeniedByStorageFilter)

## [ID3D11InfoQueue::GetNumMessagesDiscardedByMessageCountLimit](#)

Get the number of messages that were discarded due to the message count limit.  
(ID3D11InfoQueue.GetNumMessagesDiscardedByMessageCountLimit)

## [ID3D11InfoQueue::GetNumStoredMessages](#)

Get the number of messages currently stored in the message queue.  
(ID3D11InfoQueue.GetNumStoredMessages)

## [ID3D11InfoQueue::GetNumStoredMessagesAllowedByRetrievalFilter](#)

Get the number of messages that are able to pass through a retrieval filter.  
(ID3D11InfoQueue.GetNumStoredMessagesAllowedByRetrievalFilter)

## [ID3D11InfoQueue::GetRetrievalFilter](#)

Get the retrieval filter at the top of the retrieval-filter stack. (ID3D11InfoQueue.GetRetrievalFilter)

## [ID3D11InfoQueue::GetRetrievalFilterStackSize](#)

Get the size of the retrieval-filter stack in bytes. (ID3D11InfoQueue.GetRetrievalFilterStackSize)

## [ID3D11InfoQueue::GetStorageFilter](#)

Get the storage filter at the top of the storage-filter stack. (ID3D11InfoQueue.GetStorageFilter)

## [ID3D11InfoQueue::GetStorageFilterStackSize](#)

Get the size of the storage-filter stack in bytes. (ID3D11InfoQueue.GetStorageFilterStackSize)

## [ID3D11InfoQueue::PopRetrievalFilter](#)

Pop a retrieval filter from the top of the retrieval-filter stack. (ID3D11InfoQueue.PopRetrievalFilter)

## [ID3D11InfoQueue::PopStorageFilter](#)

Pop a storage filter from the top of the storage-filter stack. (ID3D11InfoQueue.PopStorageFilter)

## [ID3D11InfoQueue::PushCopyOfRetrievalFilter](#)

Push a copy of retrieval filter currently on the top of the retrieval-filter stack onto the retrieval-filter stack. (ID3D11InfoQueue.PushCopyOfRetrievalFilter)

## [ID3D11InfoQueue::PushCopyOfStorageFilter](#)

Push a copy of storage filter currently on the top of the storage-filter stack onto the storage-filter stack. (ID3D11InfoQueue.PushCopyOfStorageFilter)

### [ID3D11InfoQueue::PushEmptyRetrievalFilter](#)

Push an empty retrieval filter onto the retrieval-filter stack.  
(ID3D11InfoQueue.PushEmptyRetrievalFilter)

### [ID3D11InfoQueue::PushEmptyStorageFilter](#)

Push an empty storage filter onto the storage-filter stack.  
(ID3D11InfoQueue.PushEmptyStorageFilter)

### [ID3D11InfoQueue::PushRetrievalFilter](#)

Push a retrieval filter onto the retrieval-filter stack. (ID3D11InfoQueue.PushRetrievalFilter)

### [ID3D11InfoQueue::PushStorageFilter](#)

Push a storage filter onto the storage-filter stack. (ID3D11InfoQueue.PushStorageFilter)

### [ID3D11InfoQueue::SetBreakOnCategory](#)

Set a message category to break on when a message with that category passes through the storage filter. (ID3D11InfoQueue.SetBreakOnCategory)

### [ID3D11InfoQueue::SetBreakOnID](#)

Set a message identifier to break on when a message with that identifier passes through the storage filter. (ID3D11InfoQueue.SetBreakOnID)

### [ID3D11InfoQueue::SetBreakOnSeverity](#)

Set a message severity level to break on when a message with that severity level passes through the storage filter. (ID3D11InfoQueue.SetBreakOnSeverity)

### [ID3D11InfoQueue::SetMessageCountLimit](#)

Set the maximum number of messages that can be added to the message queue.  
(ID3D11InfoQueue.SetMessageCountLimit)

### [ID3D11InfoQueue::SetMuteDebugOutput](#)

Set a boolean that turns the debug output on or off. (ID3D11InfoQueue.SetMuteDebugOutput)

## Remarks

To get this interface, turn on debug layer and use [IUnknown::QueryInterface](#) from the [ID3D11Device](#).

**Windows Phone 8:** This API is supported.

## Requirements

<b>Minimum supported client</b>	Windows 7 [desktop apps   UWP apps]
<b>Minimum supported server</b>	Windows Server 2008 R2 [desktop apps   UWP apps]
<b>Target Platform</b>	Windows
<b>Header</b>	d3d11sdklayers.h

## See also

[IUnknown](#)

[Layer Interfaces](#)

---

## Feedback

Was this page helpful?



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

# ID3D11InfoQueue::AddApplicationMessage method (d3d11sdklayers.h)

Article 02/22/2024

Add a user-defined message to the message queue and send that message to debug output.

## Syntax

C++

```
HRESULT AddApplicationMessage(
    [in] D3D11_MESSAGE_SEVERITY Severity,
    [in] LPCSTR             pDescription
);
```

## Parameters

[in] Severity

Type: [D3D11\\_MESSAGE\\_SEVERITY](#)

Severity of a message (see [D3D11\\_MESSAGE\\_SEVERITY](#)).

[in] pDescription

Type: [LPCSTR](#)

Message string.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::AddMessage method (d3d11sdklayers.h)

Article 02/22/2024

Add a debug message to the message queue and send that message to debug output.

## Syntax

C++

```
HRESULT AddMessage(
    [in] D3D11_MESSAGE_CATEGORY Category,
    [in] D3D11_MESSAGE_SEVERITY Severity,
    [in] D3D11_MESSAGE_ID        ID,
    [in] LPCSTR                 pDescription
);
```

## Parameters

[in] Category

Type: [D3D11\\_MESSAGE\\_CATEGORY](#)

Category of a message (see [D3D11\\_MESSAGE\\_CATEGORY](#)).

[in] Severity

Type: [D3D11\\_MESSAGE\\_SEVERITY](#)

Severity of a message (see [D3D11\\_MESSAGE\\_SEVERITY](#)).

[in] ID

Type: [D3D11\\_MESSAGE\\_ID](#)

Unique identifier of a message (see [D3D11\\_MESSAGE\\_ID](#)).

[in] pDescription

Type: [LPCSTR](#)

User-defined message.

# Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 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 [ID3D11InfoQueue::AddApplicationMessage](#).

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::AddRetrievalFilterEntries method (d3d11sdklayers.h)

Article 02/22/2024

Add storage filters to the top of the retrieval-filter stack.

## Syntax

C++

```
HRESULT AddRetrievalFilterEntries(
    [in] D3D11_INFO_QUEUE_FILTER *pFilter
);
```

## Parameters

[in] pFilter

Type: [D3D11\\_INFO\\_QUEUE\\_FILTER\\*](#)

Array of retrieval filters (see [D3D11\\_INFO\\_QUEUE\\_FILTER](#)).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

The following code example shows how to use  
`ID3D11InfoQueue::AddRetrievalFilterEntries`:

```
D3D11_MESSAGE_CATEGORY cats[] = { ..., ..., ... };
D3D11_MESSAGE_SEVERITY sevs[] = { ..., ..., ... };
UINT ids[] = { ..., ..., ... };

D3D11_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(D3D11_MESSAGE_CATEGORY));
filter.AllowList.pCategoryList = cats;
filter.AllowList.NumSeverities = sizeof(sevs /
sizeof(D3D11_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

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::AddStorageFilterEntries method (d3d11sdklayers.h)

Article 02/22/2024

Add storage filters to the top of the storage-filter stack.

## Syntax

C++

```
HRESULT AddStorageFilterEntries(
    [in] D3D11_INFO_QUEUE_FILTER *pFilter
);
```

## Parameters

[in] pFilter

Type: [D3D11\\_INFO\\_QUEUE\\_FILTER\\*](#)

Array of storage filters (see [D3D11\\_INFO\\_QUEUE\\_FILTER](#)).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?



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

# ID3D11InfoQueue::ClearRetrievalFilter method (d3d11sdklayers.h)

Article02/22/2024

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

## Syntax

C++

```
void ClearRetrievalFilter();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::ClearStorageFilter method (d3d11sdklayers.h)

Article02/22/2024

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

## Syntax

C++

```
void ClearStorageFilter();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::ClearStoredMessages method (d3d11sdklayers.h)

Article 02/22/2024

Clear all messages from the message queue.

## Syntax

C++

```
void ClearStoredMessages();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetBreakOnCategory method (d3d11sdklayers.h)

Article 02/22/2024

Get a message category to break on when a message with that category passes through the storage filter.

## Syntax

C++

```
BOOL GetBreakOnCategory(  
    [in] D3D11_MESSAGE_CATEGORY Category  
) ;
```

## Parameters

[in] Category

Type: [D3D11\\_MESSAGE\\_CATEGORY](#)

Message category to break on (see [D3D11\\_MESSAGE\\_CATEGORY](#)).

## Return value

Type: [BOOL](#)

Whether this breaking condition is turned on or off (true for on, false for off).

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?



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

# ID3D11InfoQueue::GetBreakOnID method (d3d11sdklayers.h)

Article 02/22/2024

Get a message identifier to break on when a message with that identifier passes through the storage filter.

## Syntax

C++

```
BOOL GetBreakOnID(  
    [in] D3D11_MESSAGE_ID ID  
);
```

## Parameters

[in] ID

Type: [D3D11\\_MESSAGE\\_ID](#)

Message identifier to break on (see [D3D11\\_MESSAGE\\_ID](#)).

## Return value

Type: [BOOL](#)

Whether this breaking condition is turned on or off (true for on, false for off).

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?



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

# ID3D11InfoQueue::GetBreakOnSeverity method (d3d11sdklayers.h)

Article 02/22/2024

Get a message severity level to break on when a message with that severity level passes through the storage filter.

## Syntax

C++

```
BOOL GetBreakOnSeverity(  
    [in] D3D11_MESSAGE_SEVERITY Severity  
);
```

## Parameters

[in] Severity

Type: [D3D11\\_MESSAGE\\_SEVERITY](#)

Message severity level to break on (see [D3D11\\_MESSAGE\\_SEVERITY](#)).

## Return value

Type: [BOOL](#)

Whether this breaking condition is turned on or off (true for on, false for off).

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?



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

# ID3D11InfoQueue::GetMessage method (d3d11sdklayers.h)

Article 02/22/2024

Get a message from the message queue.

## Syntax

C++

```
HRESULT GetMessage(
    [in]          UINT64      MessageIndex,
    [out, optional] D3D11_MESSAGE *pMessage,
    [in, out]       SIZE_T      *pMessageByteLength
);
```

## Parameters

[in] MessageIndex

Type: [UINT64](#)

Index into message queue after an optional retrieval filter has been applied. This can be between 0 and the number of messages in the message queue that pass through the retrieval filter (which can be obtained with

[ID3D11InfoQueue::GetNumStoredMessagesAllowedByRetrievalFilter](#)). 0 is the message at the front of the message queue.

[out, optional] pMessage

Type: [D3D11\\_MESSAGE\\*](#)

Returned message (see [D3D11\\_MESSAGE](#)).

[in, out] pMessageByteLength

Type: [SIZE\\_T\\*](#)

Size of pMessage in bytes, including the size of the message string that the pMessage points to.

# Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 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
D3D11_MESSAGE * pMessage = (D3D11_MESSAGE*)malloc(messageLength);
hr = pInfoQueue->GetMessage(0, pMessage, &messageLength);
```

For an overview see [Information Queue Overview](#).

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetMessageCountLimit method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
UINT64 GetMessageCountLimit();
```

## Return value

Type: [UINT64](#)

Maximum number of messages that can be added to the queue. -1 means no limit.

## Remarks

When the number of messages in the message queue has reached the maximum limit, new messages coming in will push old messages out.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetMuteDebugOutput method (d3d11sdklayers.h)

Article 02/22/2024

Get a boolean that turns the debug output on or off.

## Syntax

C++

```
BOOL GetMuteDebugOutput();
```

## Return value

Type: **BOOL**

Whether the debug output is on or off (true for on, false for off).

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# ID3D11InfoQueue::GetNumMessagesAllowedByStorageFilter method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
UINT64 GetNumMessagesAllowedByStorageFilter();
```

## Return value

Type: [UINT64](#)

Number of messages allowed by a storage filter.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetNumMessagesDeniedByStorageFilter method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
UINT64 GetNumMessagesDeniedByStorageFilter();
```

## Return value

Type: [UINT64](#)

Number of messages denied by a storage filter.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetNumMessagesDiscardedByMessageCountLimit method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
UINT64 GetNumMessagesDiscardedByMessageCountLimit();
```

## Return value

Type: [UINT64](#)

Number of messages discarded.

## Remarks

Get and set the message count limit with [ID3D11InfoQueue::GetMessageCountLimit](#) and [ID3D11InfoQueue::SetMessageCountLimit](#), respectively.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetNumStoredMessages method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
UINT64 GetNumStoredMessages();
```

## Return value

Type: [UINT64](#)

Number of messages currently stored in the message queue.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# ID3D11InfoQueue::GetNumStoredMessagesAllowedByRetrievalFilter method (d3d11sdklayers.h)

Article 02/22/2024

Get the number of messages that are able to pass through a retrieval filter.

## Syntax

C++

```
UINT64 GetNumStoredMessagesAllowedByRetrievalFilter();
```

## Return value

Type: [UINT64](#)

Number of messages allowed by a retrieval filter.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetRetrievalFilter method (d3d11sdklayers.h)

Article 02/22/2024

Get the retrieval filter at the top of the retrieval-filter stack.

## Syntax

C++

```
HRESULT GetRetrievalFilter(
    [out, optional] D3D11_INFO_QUEUE_FILTER *pFilter,
    [in, out]      SIZE_T                 *pFilterByteLength
);
```

## Parameters

[out, optional] pFilter

Type: [D3D11\\_INFO\\_QUEUE\\_FILTER\\*](#)

Retrieval filter at the top of the retrieval-filter stack.

[in, out] pFilterByteLength

Type: [SIZE\\_T\\*](#)

Size of the retrieval filter in bytes. If pFilter is **NULL**, the size of the retrieval filter will be output to this parameter.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetRetrievalFilterStackSize method (d3d11sdklayers.h)

Article 02/22/2024

Get the size of the retrieval-filter stack in bytes.

## Syntax

C++

```
UINT GetRetrievalFilterStackSize();
```

## Return value

Type: [UINT](#)

Size of the retrieval-filter stack in bytes.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# ID3D11InfoQueue::GetStorageFilter method (d3d11sdklayers.h)

Article 02/22/2024

Get the storage filter at the top of the storage-filter stack.

## Syntax

C++

```
HRESULT GetStorageFilter(
    [out, optional] D3D11_INFO_QUEUE_FILTER *pFilter,
    [in, out]      SIZE_T                 *pFilterByteLength
);
```

## Parameters

[out, optional] pFilter

Type: [D3D11\\_INFO\\_QUEUE\\_FILTER\\*](#)

Storage filter at the top of the storage-filter stack.

[in, out] pFilterByteLength

Type: [SIZE\\_T\\*](#)

Size of the storage filter in bytes. If pFilter is **NULL**, the size of the storage filter will be output to this parameter.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::GetStorageFilterStack Size method (d3d11sdklayers.h)

Article 02/22/2024

Get the size of the storage-filter stack in bytes.

## Syntax

C++

```
UINT GetStorageFilterStackSize();
```

## Return value

Type: [UINT](#)

Size of the storage-filter stack in bytes.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# ID3D11InfoQueue::PopRetrievalFilter method (d3d11sdklayers.h)

Article02/22/2024

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

## Syntax

C++

```
void PopRetrievalFilter();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::PopStorageFilter method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
void PopStorageFilter();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::PushCopyOfRetrievalFilter method (d3d11sdklayers.h)

Article 02/22/2024

Push a copy of retrieval filter currently on the top of the retrieval-filter stack onto the retrieval-filter stack.

## Syntax

C++

```
HRESULT PushCopyOfRetrievalFilter();
```

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11InfoQueue::PushCopyOfStorageFilter method (d3d11sdklayers.h)

Article 02/22/2024

Push a copy of storage filter currently on the top of the storage-filter stack onto the storage-filter stack.

## Syntax

C++

```
HRESULT PushCopyOfStorageFilter();
```

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11InfoQueue::PushEmptyRetrievalFilter method (d3d11sdklayers.h)

Article 02/22/2024

Push an empty retrieval filter onto the retrieval-filter stack.

## Syntax

C++

```
HRESULT PushEmptyRetrievalFilter();
```

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An empty retrieval filter allows all messages to pass through.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::PushEmptyStorageFilter method (d3d11sdklayers.h)

Article 02/22/2024

Push an empty storage filter onto the storage-filter stack.

## Syntax

C++

```
HRESULT PushEmptyStorageFilter();
```

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An empty storage filter allows all messages to pass through.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::PushRetrievalFilter method (d3d11sdklayers.h)

Article 02/22/2024

Push a retrieval filter onto the retrieval-filter stack.

## Syntax

C++

```
HRESULT PushRetrievalFilter(  
    [in] D3D11_INFO_QUEUE_FILTER *pFilter  
);
```

## Parameters

[in] pFilter

Type: [D3D11\\_INFO\\_QUEUE\\_FILTER\\*](#)

Pointer to a retrieval filter (see [D3D11\\_INFO\\_QUEUE\\_FILTER](#)).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?



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

# ID3D11InfoQueue::PushStorageFilter method (d3d11sdklayers.h)

Article 02/22/2024

Push a storage filter onto the storage-filter stack.

## Syntax

C++

```
HRESULT PushStorageFilter(  
    [in] D3D11_INFO_QUEUE_FILTER *pFilter  
);
```

## Parameters

[in] pFilter

Type: [D3D11\\_INFO\\_QUEUE\\_FILTER\\*](#)

Pointer to a storage filter (see [D3D11\\_INFO\\_QUEUE\\_FILTER](#)).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?



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

# ID3D11InfoQueue::SetBreakOnCategory method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
HRESULT SetBreakOnCategory(
    [in] D3D11_MESSAGE_CATEGORY Category,
    [in] BOOL                 bEnable
);
```

## Parameters

[in] Category

Type: [D3D11\\_MESSAGE\\_CATEGORY](#)

Message category to break on (see [D3D11\\_MESSAGE\\_CATEGORY](#)).

[in] bEnable

Type: [BOOL](#)

Turns this breaking condition on or off (true for on, false for off).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::SetBreakOnID method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
HRESULT SetBreakOnID(  
    [in] D3D11_MESSAGE_ID ID,  
    [in] BOOL             bEnable  
);
```

## Parameters

[in] ID

Type: [D3D11\\_MESSAGE\\_ID](#)

Message identifier to break on (see [D3D11\\_MESSAGE\\_ID](#)).

[in] bEnable

Type: [BOOL](#)

Turns this breaking condition on or off (true for on, false for off).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::SetBreakOnSeverity method (d3d11sdklayers.h)

Article 02/22/2024

Set a message severity level to break on when a message with that severity level passes through the storage filter.

## Syntax

C++

```
HRESULT SetBreakOnSeverity(
    [in] D3D11_MESSAGE_SEVERITY Severity,
    [in] BOOL             bEnable
);
```

## Parameters

[in] Severity

Type: [D3D11\\_MESSAGE\\_SEVERITY](#)

A [D3D11\\_MESSAGE\\_SEVERITY](#), which represents a message severity level to break on.

[in] bEnable

Type: [BOOL](#)

Turns this breaking condition on or off (true for on, false for off).

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::SetMessageCountLimit method (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
HRESULT SetMessageCountLimit(  
    [in] UINT64 MessageCountLimit  
);
```

## Parameters

[in] MessageCountLimit

Type: [UINT64](#)

Maximum number of messages that can be added to the message queue. -1 means no limit.

## Return value

Type: [HRESULT](#)

This method returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

When the number of messages in the message queue has reached the maximum limit, new messages coming in will push old messages out.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11InfoQueue::SetMuteDebugOutput method (d3d11sdklayers.h)

Article 02/22/2024

Set a boolean that turns the debug output on or off.

## Syntax

C++

```
void SetMuteDebugOutput(  
    [in] BOOL bMute  
);
```

## Parameters

[in] bMute

Type: **BOOL**

Disable/Enable the debug output (**TRUE** to disable or mute the output, **FALSE** to enable the output).

## Return value

None

## Remarks

This will stop messages that pass the storage filter from being printed out in the debug output, however those messages will still be added to the message queue.

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3D11.lib

## See also

[ID3D11InfoQueue Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RefDefaultTrackingOptions interface (d3d11sdklayers.h)

Article 02/22/2024

The default tracking interface sets reference default tracking options.

## Inheritance

The **ID3D11RefDefaultTrackingOptions** interface inherits from the [IUnknown](#) interface. **ID3D11RefDefaultTrackingOptions** also has these types of members:

## Methods

The **ID3D11RefDefaultTrackingOptions** interface has these methods.

[+] Expand table

<p><a href="#">ID3D11RefDefaultTrackingOptions::SetTrackingOptions</a></p> <p>Sets graphics processing unit (GPU) debug reference default tracking options for specific resource types.</p>

## Remarks

These APIs require the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows

Requirement	Value
Header	d3d11sdklayers.h

## See also

[IUnknown](#)

[Layer Interfaces](#)

---

## Feedback

Was this page helpful?



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

# ID3D11RefDefaultTrackingOptions::SetTrackingOptions method (d3d11sdklayers.h)

Article 02/22/2024

Sets graphics processing unit (GPU) debug reference default tracking options for specific resource types.

## Syntax

C++

```
HRESULT SetTrackingOptions(  
    UINT ResourceTypeFlags,  
    UINT Options  
);
```

## Parameters

`ResourceTypeFlags`

A [D3D11\\_SHADER\\_TRACKING\\_RESOURCE\\_TYPE](#)-typed value that specifies the type of resource to track.

`Options`

A combination of [D3D11\\_SHADER\\_TRACKING\\_OPTIONS](#)-typed flags that are combined by using a bitwise **OR** operation. The resulting value identifies tracking options. If a flag is present, the tracking option that the flag represents is set to "on"; otherwise the tracking option is set to "off."

## Return value

This method returns one of the [Direct3D 11 return codes](#).

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3DCompiler.lib

## See also

[ID3D11RefDefaultTrackingOptions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RefTrackingOptions interface (d3d11sdklayers.h)

Article 02/22/2024

The tracking interface sets reference tracking options.

## Inheritance

The **ID3D11RefTrackingOptions** interface inherits from the [IUnknown](#) interface.

**ID3D11RefTrackingOptions** also has these types of members:

## Methods

The **ID3D11RefTrackingOptions** interface has these methods.

[+] Expand table

<p><a href="#">ID3D11RefTrackingOptions::SetTrackingOptions</a></p> <p>Sets graphics processing unit (GPU) debug reference tracking options.</p>

## Remarks

These APIs require the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11sdklayers.h

## See also

IUnknown

[Layer Interfaces](#)

---

## Feedback

Was this page helpful?



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

# ID3D11RefTrackingOptions::SetTrackingOptions method (d3d11sdklayers.h)

Article 02/22/2024

Sets graphics processing unit (GPU) debug reference tracking options.

## Syntax

C++

```
HRESULT SetTrackingOptions(  
    UINT uOptions  
);
```

## Parameters

`uOptions`

A combination of [D3D11\\_SHADER\\_TRACKING\\_OPTIONS](#)-typed flags that are combined by using a bitwise **OR** operation. The resulting value identifies tracking options. If a flag is present, the tracking option that the flag represents is set to "on"; otherwise the tracking option is set to "off."

## Return value

This method returns one of the [Direct3D 11 return codes](#).

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3DCompiler.lib

## See also

[ID3D11RefTrackingOptions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11SwitchToRef interface (d3d11sdklayers.h)

Article02/22/2024

**Note** The ID3D11SwitchToRef interface and its methods are not supported in Direct3D 11.

## Inheritance

The ID3D11SwitchToRef interface inherits from the [IUnknown](#) interface.

ID3D11SwitchToRef also has these types of members:

## Methods

The ID3D11SwitchToRef interface has these methods.

[+] Expand table

<a href="#">ID3D11SwitchToRef::GetUseRef</a>
ID3D11SwitchToRef::GetUseRef method
<a href="#">ID3D11SwitchToRef::SetUseRef</a>
ID3D11SwitchToRef::SetUseRef method

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h

## See also

[IUnknown](#)

[Layer Interfaces](#)

---

## Feedback

Was this page helpful?



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

# ID3D11SwitchToRef::GetUseRef method (d3d11sdklayers.h)

Article02/22/2024

**Note** The [ID3D11SwitchToRef](#) interface and its methods are not supported in Direct3D 11.

## Syntax

C++

```
BOOL GetUseRef();
```

## Return value

Type: [BOOL](#)

Reserved.

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11sdklayers.h

## See also

[ID3D11SwitchToRef Interface](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11SwitchToRef::SetUseRef method (d3d11sdklayers.h)

Article02/22/2024

**Note** The [ID3D11SwitchToRef](#) interface and its methods are not supported in Direct3D 11.

## Syntax

C++

```
BOOL SetUseRef(  
    BOOL UseRef  
) ;
```

## Parameters

UseRef

Type: [BOOL](#)

Reserved.

## Return value

Type: [BOOL](#)

Reserved.

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d11sdklayers.h

## See also

[ID3D11SwitchToRef Interface](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# ID3D11TracingDevice interface (d3d11sdklayers.h)

Article 02/22/2024

The tracing device interface sets shader tracking information, which enables accurate logging and playback of shader execution.

## Inheritance

The **ID3D11TracingDevice** interface inherits from the [IUnknown](#) interface.

**ID3D11TracingDevice** also has these types of members:

## Methods

The **ID3D11TracingDevice** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11TracingDevice::SetShaderTrackingOptions</a>
Sets the reference rasterizer's race-condition tracking options for a specific shader.

<a href="#">ID3D11TracingDevice::SetShaderTrackingOptionsByType</a>
Sets the reference rasterizer's default race-condition tracking options for the specified resource types.

## Remarks

To get this interface, turn on the [debug layer](#) and use [IUnknown::QueryInterface](#) from the [ID3D11Device](#).

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11sdklayers.h

## See also

[IUnknown](#)

[Layer Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11TracingDevice::SetShaderTrackingOptions method (d3d11sdklayers.h)

Article 02/22/2024

Sets the reference rasterizer's race-condition tracking options for a specific shader.

## Syntax

C++

```
HRESULT SetShaderTrackingOptions(
    [in] IUnknown *pShader,
    [in] UINT     Options
);
```

## Parameters

[in] pShader

A pointer to the [IUnknown](#) interface of a shader.

[in] Options

A combination of [D3D11\\_SHADER\\_TRACKING\\_OPTIONS](#)-typed flags that are combined by using a bitwise **OR** operation. The resulting value identifies tracking options. If a flag is present, the tracking option that the flag represents is set to "on"; otherwise the tracking option is set to "off."

## Return value

This method returns one of the [Direct3D 11 return codes](#).

## Remarks

[D3D11\\_SHADER\\_TRACKING\\_OPTION\\_UAV\\_SPECIFIC\\_FLAGS](#)) in the *Options* parameter for a compute shader, **SetShaderTrackingOptions** ignores it.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

# Requirements

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

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3DCompiler.lib

## See also

[ID3D11TracingDevice](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11TracingDevice::SetShaderTrackingOptionsByType method (d3d11sdklayers.h)

Article 02/22/2024

Sets the reference rasterizer's default race-condition tracking options for the specified resource types.

## Syntax

C++

```
HRESULT SetShaderTrackingOptionsByType(
    [in] UINT ResourceTypeFlags,
    [in] UINT Options
);
```

## Parameters

[in] ResourceTypeFlags

A [D3D11\\_SHADER\\_TRACKING\\_RESOURCE\\_TYPE](#)-typed value that specifies the type of resource to track.

[in] Options

A combination of [D3D11\\_SHADER\\_TRACKING\\_OPTIONS](#)-typed flags that are combined by using a bitwise **OR** operation. The resulting value identifies tracking options. If a flag is present, the tracking option that the flag represents is set to "on," otherwise the tracking option is set to "off."

## Return value

This method returns one of the [Direct3D 11 return codes](#).

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11sdklayers.h
Library	D3DCompiler.lib

## See also

[ID3D11TracingDevice](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Layer Structures

Article • 11/04/2020

This section contains information about the layer structures.

## In this section

Topic	Description
<a href="#">D3D11_INFO_QUEUE_FILTER</a>	Debug message filter; contains a lists of message types to allow or deny.
<a href="#">D3D11_INFO_QUEUE_FILTER_DESC</a>	Allow or deny certain types of messages to pass through a filter.
<a href="#">D3D11_MESSAGE</a>	A debug message in the Information Queue.

## Related topics

[Layer Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_INFO\_QUEUE\_FILTER structure (d3d11sdklayers.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_INFO_QUEUE_FILTER {
    D3D11_INFO_QUEUE_FILTER_DESC AllowList;
    D3D11_INFO_QUEUE_FILTER_DESC DenyList;
} D3D11_INFO_QUEUE_FILTER;
```

## Members

AllowList

Type: [D3D11\\_INFO\\_QUEUE\\_FILTER\\_DESC](#)

Types of messages that you want to allow. See [D3D11\\_INFO\\_QUEUE\\_FILTER\\_DESC](#).

DenyList

Type: [D3D11\\_INFO\\_QUEUE\\_FILTER\\_DESC](#)

Types of messages that you want to deny.

## Remarks

For use with an [ID3D11InfoQueue](#) Interface.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11sdklayers.h

## See also

[Core Structures](#)

[Layer Structures](#)

---

## Feedback

Was this page helpful?



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

# D3D11\_INFO\_QUEUE\_FILTER\_DESC structure (d3d11sdklayers.h)

Article 07/27/2022

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

## Syntax

C++

```
typedef struct D3D11_INFO_QUEUE_FILTER_DESC {
    UINT NumCategories;
    D3D11_MESSAGE_CATEGORY *pCategoryList;
    UINT NumSeverities;
    D3D11_MESSAGE_SEVERITY *pSeverityList;
    UINT NumIDs;
    D3D11_MESSAGE_ID *pIDList;
} D3D11_INFO_QUEUE_FILTER_DESC;
```

## Members

NumCategories

Type: [UINT](#)

Number of message categories to allow or deny.

pCategoryList

Type: [D3D11\\_MESSAGE\\_CATEGORY\\*](#)

Array of message categories to allow or deny. Array must have at least NumCategories members (see [D3D11\\_MESSAGE\\_CATEGORY](#)).

NumSeverities

Type: [UINT](#)

Number of message severity levels to allow or deny.

pSeverityList

Type: [D3D11\\_MESSAGE\\_SEVERITY\\*](#)

Array of message severity levels to allow or deny. Array must have at least NumSeverities members (see [D3D11\\_MESSAGE\\_SEVERITY](#)).

NumIDs

Type: [UINT](#)

Number of message IDs to allow or deny.

pIDList

Type: [D3D11\\_MESSAGE\\_ID\\*](#)

Array of message IDs to allow or deny. Array must have at least NumIDs members (see [D3D11\\_MESSAGE\\_ID](#)).

## Requirements

Header	d3d11sdklayers.h
--------	------------------

## See also

[Core Structures](#)

[Layer Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_MESSAGE structure (d3d11sdklayers.h)

Article 02/22/2024

A debug message in the Information Queue.

## Syntax

C++

```
typedef struct D3D11_MESSAGE {
    D3D11_MESSAGE_CATEGORY Category;
    D3D11_MESSAGE_SEVERITY Severity;
    D3D11_MESSAGE_ID       ID;
    const char              *pDescription;
    SIZE_T                  DescriptionByteLength;
} D3D11_MESSAGE;
```

## Members

Category

Type: [D3D11\\_MESSAGE\\_CATEGORY](#)

The category of the message. See [D3D11\\_MESSAGE\\_CATEGORY](#).

Severity

Type: [D3D11\\_MESSAGE\\_SEVERITY](#)

The severity of the message. See [D3D11\\_MESSAGE\\_SEVERITY](#).

ID

Type: [D3D11\\_MESSAGE\\_ID](#)

The ID of the message. See [D3D11\\_MESSAGE\\_ID](#).

pDescription

Type: [const char\\*](#)

The message string.

DescriptionByteLength

Type: [SIZE\\_T](#)

The length of pDescription in bytes.

## Remarks

This structure is returned from [ID3D11InfoQueue::GetMessage](#) as part of the Information Queue feature (see [ID3D11InfoQueue Interface](#)).

## Requirements

[Expand table](#)

Requirement	Value
Header	d3d11sdklayers.h

## See also

[Core Structures](#)

[Layer Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Layer Enumerations

Article • 11/04/2020

This section contains information about the layer enumerations.

## In this section

Topic	Description
<a href="#">D3D11_MESSAGE_CATEGORY</a>	Categories of debug messages. This will identify the category of a message when retrieving a message with <a href="#">ID3D11InfoQueue::GetMessage</a> and when adding a message with <a href="#">ID3D11InfoQueue::AddMessage</a> . When creating an <a href="#">info queue filter</a> , these values can be used to allow or deny any categories of messages to pass through the storage and retrieval filters.
<a href="#">D3D11_MESSAGE_ID</a>	Debug messages for setting up an info-queue filter (see <a href="#">D3D11_INFO_QUEUE_FILTER</a> ); 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 <a href="#">ID3D11InfoQueue::GetMessage</a> or <a href="#">ID3D11InfoQueue::AddMessage</a> .
<a href="#">D3D11_MESSAGE_SEVERITY</a>	Debug message severity levels for an information queue.
<a href="#">D3D11_RLDO_FLAGS</a>	Options for the amount of information to report about a device object's lifetime.
<a href="#">D3D11_SHADER_TRACKING_OPTIONS</a>	Options that specify how to perform shader debug tracking.
<a href="#">D3D11_SHADER_TRACKING_RESOURCE_TYPE</a>	Indicates which resource types to track.

## Related topics

[Layer Reference](#)

## Feedback

Was this page helpful?  

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

# D3D11\_MESSAGE\_CATEGORY enumeration (d3d11sdklayers.h)

Article 02/22/2024

Categories of debug messages. This will identify the category of a message when retrieving a message with [ID3D11InfoQueue::GetMessage](#) and when adding a message with [ID3D11InfoQueue::AddMessage](#). When creating an [info queue filter](#), these values can be used to allow or deny any categories of messages to pass through the storage and retrieval filters.

## Syntax

C++

```
typedef enum D3D11_MESSAGE_CATEGORY {
    D3D11_MESSAGE_CATEGORY_APPLICATION_DEFINED = 0,
    D3D11_MESSAGE_CATEGORY_MISCELLANEOUS,
    D3D11_MESSAGE_CATEGORY_INITIALIZATION,
    D3D11_MESSAGE_CATEGORY_CLEANUP,
    D3D11_MESSAGE_CATEGORY_COMPILATION,
    D3D11_MESSAGE_CATEGORY_STATE_CREATION,
    D3D11_MESSAGE_CATEGORY_STATE_SETTING,
    D3D11_MESSAGE_CATEGORY_STATE_GETTING,
    D3D11_MESSAGE_CATEGORY_RESOURCE_MANIPULATION,
    D3D11_MESSAGE_CATEGORY_EXECUTION,
    D3D11_MESSAGE_CATEGORY_SHADER
};
```

## Constants

[+] Expand table

D3D11\_MESSAGE\_CATEGORY\_APPLICATION\_DEFINED

Value: 0

User defined message. See [ID3D11InfoQueue::AddMessage](#).

D3D11\_MESSAGE\_CATEGORY\_MISCELLANEOUS

D3D11\_MESSAGE\_CATEGORY\_INITIALIZATION

D3D11\_MESSAGE\_CATEGORY\_CLEANUP

D3D11\_MESSAGE\_CATEGORY\_COMPILATION

D3D11\_MESSAGE\_CATEGORY\_STATE\_CREATION

D3D11\_MESSAGE\_CATEGORY\_STATE\_SETTING

D3D11\_MESSAGE\_CATEGORY\_STATE\_GETTING

D3D11\_MESSAGE\_CATEGORY\_RESOURCE\_MANIPULATION

D3D11\_MESSAGE\_CATEGORY\_EXECUTION

D3D11\_MESSAGE\_CATEGORY\_SHADER

Direct3D 11: This value is not supported until Direct3D 11.1.

## Remarks

This is part of the Information Queue feature. See [ID3D11InfoQueue Interface](#).

## Requirements

[ ] Expand table

Requirement	Value
Header	d3d11sdklayers.h

## See also

[Layer Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_MESSAGE\_ID enumeration (d3d11sdklayers.h)

Article 05/24/2022

Debug messages for setting up an info-queue filter (see [D3D11\\_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 [ID3D11InfoQueue::GetMessage](#) or [ID3D11InfoQueue::AddMessage](#).

## Syntax

C++

```
typedef enum D3D11_MESSAGE_ID {
    D3D11_MESSAGE_ID_UNKNOWN = 0,
    D3D11_MESSAGE_ID_DEVICE_IASETVERTEXBUFFERS_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_IASETINDEXBUFFER_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_VSSETSHADERRESOURCES_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_VSSETCONSTANTBUFFERS_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_GSSETSHADERRESOURCES_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_GSSETCONSTANTBUFFERS_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_PSSETSHADERRESOURCES_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_PSSETCONSTANTBUFFERS_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_OMSETRENDERTARGETS_HAZARD,
    D3D11_MESSAGE_ID_DEVICE_SOSETTARGETS_HAZARD,
    D3D11_MESSAGE_ID_STRING_FROM_APPLICATION,
    D3D11_MESSAGE_ID_CORRUPTED_THIS,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER1,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER2,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER3,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER4,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER5,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER6,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER7,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER8,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER9,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER10,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER11,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER12,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER13,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER14,
    D3D11_MESSAGE_ID_CORRUPTED_PARAMETER15,
    D3D11_MESSAGE_ID_CORRUPTED_MULTITHREADING,
    D3D11_MESSAGE_ID_MESSAGE_REPORTING_OUTOFMEMORY,
    D3D11_MESSAGE_ID_IASETINPUTLAYOUT_UNBINDDLETINGOBJECT,
    D3D11_MESSAGE_ID_IASETVERTEXBUFFERS_UNBINDDLETINGOBJECT,
    D3D11_MESSAGE_ID_IASETINDEXBUFFER_UNBINDDLETINGOBJECT,
    D3D11_MESSAGE_ID_VSSETSHADER_UNBINDDLETINGOBJECT,
```

```
D3D11_MESSAGE_ID_VSSETSHADERRESOURCES_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_VSSETCONSTANTBUFFERS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_VSSETSAMPLERS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_GSSETSHADER_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_GSSETSHADERRESOURCES_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_GSSETCONSTANTBUFFERS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_GSSETSAMPLERS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_SOSETTARGETS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_PSSETSHADER_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_PSSETSHADERRESOURCES_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_PSSETCONSTANTBUFFERS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_PSSETSAMPLERS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_RSSETSTATE_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_OMSETBLENDSTATE_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_OMSETDEPTHSTENCILSTATE_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_OMSETRENDERTARGETS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_SETPREDICATION_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_GETPRIVATEDATA_MOREDATA,
D3D11_MESSAGE_ID_SETPRIVATEDATA_INVALIDFREEDATA,
D3D11_MESSAGE_ID_SETPRIVATEDATA_INVALIDUNKNOWN,
D3D11_MESSAGE_ID_SETPRIVATEDATA_INVALIDFLAGS,
D3D11_MESSAGE_ID_SETPRIVATEDATA_CHANGINGPARAMS,
D3D11_MESSAGE_ID_SETPRIVATEDATA_OUTOFMEMORY,
D3D11_MESSAGE_ID_CREATEBUFFER_UNRECOGNIZEDFORMAT,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDDSAMPLES,
D3D11_MESSAGE_ID_CREATEBUFFER_UNRECOGNIZEDUSAGE,
D3D11_MESSAGE_ID_CREATEBUFFER_UNRECOGNIZEDBINDFLAGS,
D3D11_MESSAGE_ID_CREATEBUFFER_UNRECOGNIZEDCPUACCESSFLAGS,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDCPUACCESSFLAGS,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDDESCFLAGS,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDINITIALDATA,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDDIMENSIONS,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDMIPLEVELS,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDMISCFFLAGS,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATEBUFFER_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATEBUFFER_NULLDESC,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDCONSTANTBUFFERBINDINGS,
D3D11_MESSAGE_ID_CREATEBUFFER_LARGEALLOCATION,
D3D11_MESSAGE_ID_CREATETEXTURE1D_UNRECOGNIZEDFORMAT,
D3D11_MESSAGE_ID_CREATETEXTURE1D_UNSUPPORTEDFORMAT,
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDDSAMPLES,
D3D11_MESSAGE_ID_CREATETEXTURE1D_UNRECOGNIZEDUSAGE,
D3D11_MESSAGE_ID_CREATETEXTURE1D_UNRECOGNIZEDBINDFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE1D_UNRECOGNIZEDCPUACCESSFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE1D_UNRECOGNIZEDMISCFFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDCPUACCESSFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDBINDFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDINITIALDATA,
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDDIMENSIONS,
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDMIPLEVELS,
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDMISCFFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATETEXTURE1D_OUTOFMEMORY_RETURN,
```

```
D3D11_MESSAGE_ID_CREATETEXTURE1D_NULLDESC,
D3D11_MESSAGE_ID_CREATETEXTURE1D_LARGEALLOCATION,
D3D11_MESSAGE_ID_CREATETEXTURE2D_UNRECOGNIZEDFORMAT,
D3D11_MESSAGE_ID_CREATETEXTURE2D_UNSUPPORTEDFORMAT,
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDSAMPLES,
D3D11_MESSAGE_ID_CREATETEXTURE2D_UNRECOGNIZEDUSAGE,
D3D11_MESSAGE_ID_CREATETEXTURE2D_UNRECOGNIZEDBINDFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE2D_UNRECOGNIZEDCPUACCESSFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE2D_UNRECOGNIZEDMISCFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDCPUACCESSFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDBINDFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDINITIALDATA,
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDDIMENSIONS,
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDMIPLEVELS,
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDMISCFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATETEXTURE2D_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATETEXTURE2D_NULLDESC,
D3D11_MESSAGE_ID_CREATETEXTURE2D_LARGEALLOCATION,
D3D11_MESSAGE_ID_CREATETEXTURE3D_UNRECOGNIZEDFORMAT,
D3D11_MESSAGE_ID_CREATETEXTURE3D_UNSUPPORTEDFORMAT,
D3D11_MESSAGE_ID_CREATETEXTURE3D_INVALIDSAMPLES,
D3D11_MESSAGE_ID_CREATETEXTURE3D_UNRECOGNIZEDUSAGE,
D3D11_MESSAGE_ID_CREATETEXTURE3D_UNRECOGNIZEDBINDFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE3D_UNRECOGNIZEDCPUACCESSFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE3D_UNRECOGNIZEDMISCFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE3D_INVALIDCPUACCESSFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE3D_INVALIDBINDFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE3D_INVALIDINITIALDATA,
D3D11_MESSAGE_ID_CREATETEXTURE3D_INVALIDDIMENSIONS,
D3D11_MESSAGE_ID_CREATETEXTURE3D_INVALIDMIPLEVELS,
D3D11_MESSAGE_ID_CREATETEXTURE3D_INVALIDMISCFLAGS,
D3D11_MESSAGE_ID_CREATETEXTURE3D_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATETEXTURE3D_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATETEXTURE3D_NULLDESC,
D3D11_MESSAGE_ID_CREATETEXTURE3D_LARGEALLOCATION,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_UNRECOGNIZEDFORMAT,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDESC,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDFORMAT,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDIMENSIONS,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDRESOURCE,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_TOOMANYOBJECTS,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_UNRECOGNIZEDFORMAT,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_UNSUPPORTEDFORMAT,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDESC,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDFORMAT,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDIMENSIONS,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDRESOURCE,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_TOOMANYOBJECTS,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATERENDERTARGETVIEW_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_UNRECOGNIZEDFORMAT,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDDESC,
```

```
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDFORMAT,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDDIMENSIONS,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDRESOURCE,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_TOOMANYOBJECTS,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_OUTOFCMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_OUTOFCMEMORY,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_TOOMANYELEMENTS,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDFORMAT,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_INCOMPATIBLEFORMAT,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDDSLOT,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDINPUTSLOTCLASS,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_STEPRATESLOTCLASSMISMATCH,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDDSLOTCLASSCHANGE,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSTEPRATECHANGE,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDALIGNMENT,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_DUPLICATESEMANTIC,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_UNPARSEABLEINPUTSIGNATURE,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_NULLSEMANTIC,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_MISSINGELEMENT,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_NULLDESC,
D3D11_MESSAGE_ID_CREATEVERTEXSHADER_OUTOFCMEMORY,
D3D11_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERBYTECODE,
D3D11_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERTYPE,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADER_OUTOFCMEMORY,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERBYTECODE,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERTYPE,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTOFCMEMORY,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERBYTECODE,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERTYPE,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMENTRIES,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSTREAMSTRIDEUNUS
ED,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDDECL,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_EXPECTEDDECL,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSLOT0EXPECTED,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSLOT,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_ONLYONEELEMENTPERSLOT,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDCOMPONENTCOUNT,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTARTCOMPONENTA
NDCOMPONENTCOUNT,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDGAPDEFINITION,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_REPEATODOPUT,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSTREAMSTR
IDE,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGSEMANTIC,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MASKMISMATCH,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_CANHAVEONLYGAPS,
```

```
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DECLTOOCOMPLEX,  
,  
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGOUTPUTSIGNATURE  
,  
D3D11_MESSAGE_ID_CREATEPIXELSHADER_OUTOFCMEMORY,  
D3D11_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERBYTECODE,  
D3D11_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERTYPE,  
D3D11_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDFILLMODE,  
D3D11_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDCULLMODE,  
D3D11_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDDEPTHBIASCLAMP,  
D3D11_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDSLOPESCALEDDDEPTHBIAS,  
D3D11_MESSAGE_ID_CREATERASTERIZERSTATE_TOOMANYOBJECTS,  
D3D11_MESSAGE_ID_CREATERASTERIZERSTATE_NULLDESC,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDDEPTHWRITEMASK,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDDEPTHFUNC,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTFACE_STENCILFAILOP,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTFACE_STENCILZFAILOP,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTFACE_STENCILPASSOP,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTFACE_STENCILFUNC,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKFACE_STENCILFAILOP,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKFACE_STENCILZFAILOP,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKFACE_STENCILPASSOP,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKFACE_STENCILFUNC,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_TOOMANYOBJECTS,  
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_NULLDESC,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLEND,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLEND,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOP,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLENDALPHA,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLENDALPHA,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOPALPHA,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_INVALIDRENDERTARGETWRITEMASK,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_TOOMANYOBJECTS,  
D3D11_MESSAGE_ID_CREATEBLENDSTATE_NULLDESC,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDFILTER,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDADDRESSU,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDADDRESSV,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDADDRESSW,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDMIPLODBIAS,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDMAXANISOTROPY,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDCOMPARISONFUNC,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDMINLOD,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_INVALIDMAXLOD,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_TOOMANYOBJECTS,  
D3D11_MESSAGE_ID_CREAMPSAMPLERSTATE_NULLDESC,  
D3D11_MESSAGE_ID_CREATEQUERYORPREDICATE_INVALIDQUERY,  
D3D11_MESSAGE_ID_CREATEQUERYORPREDICATE_INVALIDMISCFLAGS,  
D3D11_MESSAGE_ID_CREATEQUERYORPREDICATE_UNEXPECTEDMISCFLAG,  
D3D11_MESSAGE_ID_CREATEQUERYORPREDICATE_NULLDESC,  
D3D11_MESSAGE_ID_DEVICE_IASETPRIMITIVETOPOLOGY_TOPOLOGY_UNRECOGNIZED,  
D3D11_MESSAGE_ID_DEVICE_IASETPRIMITIVETOPOLOGY_TOPOLOGY_UNDEFINED,  
D3D11_MESSAGE_ID_IASETVERTEXBUFFERS_INVALIDBUFFER,  
D3D11_MESSAGE_ID_DEVICE_IASETVERTEXBUFFERS_OFFSET_TOO_LARGE,  
D3D11_MESSAGE_ID_DEVICE_IASETVERTEXBUFFERS_BUFFERS_EMPTY,  
D3D11_MESSAGE_ID_IASETINDEXBUFFER_INVALIDBUFFER,
```

```
D3D11_MESSAGE_ID_DEVICE_IASETINDEXBUFFER_FORMAT_INVALID,
D3D11_MESSAGE_ID_DEVICE_IASETINDEXBUFFER_OFFSET_TOO_LARGE,
D3D11_MESSAGE_ID_DEVICE_IASETINDEXBUFFER_OFFSET_UNALIGNED,
D3D11_MESSAGE_ID_DEVICE_VSSETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_VSSETCONSTANTBUFFERS_INVALIDBUFFER,
D3D11_MESSAGE_ID_DEVICE_VSSETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_VSSETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_GSSETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_GSSETCONSTANTBUFFERS_INVALIDBUFFER,
D3D11_MESSAGE_ID_DEVICE_GSSETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_GSSETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_SOSETTARGETS_INVALIDBUFFER,
D3D11_MESSAGE_ID_DEVICE_SOSETTARGETS_OFFSET_UNALIGNED,
D3D11_MESSAGE_ID_DEVICE_PSSETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_PSSETCONSTANTBUFFERS_INVALIDBUFFER,
D3D11_MESSAGE_ID_DEVICE_PSSETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_PSSETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_RSSETPORTS_INVALIDVIEWPORT,
D3D11_MESSAGE_ID_DEVICE_RSSETSCISSORRECTS_INVALIDSCISSOR,
D3D11_MESSAGE_ID_CLEARRENDERTARGETVIEW_DENORMFLUSH,
D3D11_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_DENORMFLUSH,
D3D11_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_INVALID,
D3D11_MESSAGE_ID_DEVICE_IAGETVERTEXBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_VSGETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_VSGETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_VSGETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_GSGETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_GSGETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_GSGETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_SOGETTARGETS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_PSGETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_PSGETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_PSGETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_RSGETVIEWPORTS_VIEWPORTS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_RSGETSCISSORRECTS_RECTS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_GENERATEMIPS_RESOURCE_INVALID,
D3D11_MESSAGE_ID_COPYSUBRESOURCEREGION_INVALIDDESTINATIONSUBRESOURCE,
D3D11_MESSAGE_ID_COPYSUBRESOURCEREGION_INVALIDSOURCESUBRESOURCE,
D3D11_MESSAGE_ID_COPYSUBRESOURCEREGION_INVALIDSOURCEBOX,
D3D11_MESSAGE_ID_COPYSUBRESOURCEREGION_INVALIDSOURCE,
D3D11_MESSAGE_ID_COPYSUBRESOURCEREGION_INVALIDDESTINATIONSTATE,
D3D11_MESSAGE_ID_COPYSUBRESOURCEREGION_INVALIDSOURCESTATE,
D3D11_MESSAGE_ID_COPYRESOURCE_INVALIDDSOURCE,
D3D11_MESSAGE_ID_COPYRESOURCE_INVALIDDESTINATIONSTATE,
D3D11_MESSAGE_ID_COPYRESOURCE_INVALIDDSOURCESTATE,
D3D11_MESSAGE_ID_UPDATESUBRESOURCE_INVALIDDESTINATIONSUBRESOURCE,
D3D11_MESSAGE_ID_UPDATESUBRESOURCE_INVALIDDESTINATIONBOX,
D3D11_MESSAGE_ID_UPDATESUBRESOURCE_INVALIDDESTINATIONSTATE,
D3D11_MESSAGE_ID_DEVICE_RESOLVESUBRESOURCE_DESTINATION_INVALID,
D3D11_MESSAGE_ID_DEVICE_RESOLVESUBRESOURCE_DESTINATION_SUBRESOURCE_INVALID,
D3D11_MESSAGE_ID_DEVICE_RESOLVESUBRESOURCE_SOURCE_INVALID,
D3D11_MESSAGE_ID_DEVICE_RESOLVESUBRESOURCE_SOURCE_SUBRESOURCE_INVALID,
D3D11_MESSAGE_ID_DEVICE_RESOLVESUBRESOURCE_FORMAT_INVALID,
D3D11_MESSAGE_ID_BUFFER_MAP_INVALIDMAPTYPE,
```

```
D3D11_MESSAGE_ID_BUFFER_MAP_INVALIDFLAGS,
D3D11_MESSAGE_ID_BUFFER_MAP_ALREADYMAPPED,
D3D11_MESSAGE_ID_BUFFER_MAP_DEVICEREMOVED_RETURN,
D3D11_MESSAGE_ID_BUFFER_UNMAP_NOTMAPPED,
D3D11_MESSAGE_ID_TEXTURE1D_MAP_INVALIDMAPTYPE,
D3D11_MESSAGE_ID_TEXTURE1D_MAP_INVALIDSUBRESOURCE,
D3D11_MESSAGE_ID_TEXTURE1D_MAP_INVALIDFLAGS,
D3D11_MESSAGE_ID_TEXTURE1D_MAP_ALREADYMAPPED,
D3D11_MESSAGE_ID_TEXTURE1D_MAP_DEVICEREMOVED_RETURN,
D3D11_MESSAGE_ID_TEXTURE1D_UNMAP_INVALIDSUBRESOURCE,
D3D11_MESSAGE_ID_TEXTURE1D_UNMAP_NOTMAPPED,
D3D11_MESSAGE_ID_TEXTURE2D_MAP_INVALIDMAPTYPE,
D3D11_MESSAGE_ID_TEXTURE2D_MAP_INVALIDSUBRESOURCE,
D3D11_MESSAGE_ID_TEXTURE2D_MAP_INVALIDFLAGS,
D3D11_MESSAGE_ID_TEXTURE2D_MAP_ALREADYMAPPED,
D3D11_MESSAGE_ID_TEXTURE2D_MAP_DEVICEREMOVED_RETURN,
D3D11_MESSAGE_ID_TEXTURE2D_UNMAP_INVALIDSUBRESOURCE,
D3D11_MESSAGE_ID_TEXTURE2D_UNMAP_NOTMAPPED,
D3D11_MESSAGE_ID_TEXTURE3D_MAP_INVALIDMAPTYPE,
D3D11_MESSAGE_ID_TEXTURE3D_MAP_INVALIDSUBRESOURCE,
D3D11_MESSAGE_ID_TEXTURE3D_MAP_INVALIDFLAGS,
D3D11_MESSAGE_ID_TEXTURE3D_MAP_ALREADYMAPPED,
D3D11_MESSAGE_ID_TEXTURE3D_MAP_DEVICEREMOVED_RETURN,
D3D11_MESSAGE_ID_TEXTURE3D_UNMAP_INVALIDSUBRESOURCE,
D3D11_MESSAGE_ID_TEXTURE3D_UNMAP_NOTMAPPED,
D3D11_MESSAGE_ID_CHECKFORMATSUPPORT_FORMAT_DEPRECATED,
D3D11_MESSAGE_ID_CHECKMULTISAMPLEQUALITYLEVELS_FORMAT_DEPRECATED,
D3D11_MESSAGE_ID_SETEXCEPTIONMODE_UNRECOGNIZEDFLAGS,
D3D11_MESSAGE_ID_SETEXCEPTIONMODE_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_SETEXCEPTIONMODE_DEVICEREMOVED_RETURN,
D3D11_MESSAGE_ID_REF_SIMULATING_INFINITELY_FAST_HARDWARE,
D3D11_MESSAGE_ID_REF_THREADING_MODE,
D3D11_MESSAGE_ID_REF_UMDRIVER_EXCEPTION,
D3D11_MESSAGE_ID_REF_KMDRIVER_EXCEPTION,
D3D11_MESSAGE_ID_REF_HARDWARE_EXCEPTION,
D3D11_MESSAGE_ID_REF_ACCESSING_INDEXABLE_TEMP_OUT_OF_RANGE,
D3D11_MESSAGE_ID_REF_PROBLEM_PARSING_SHADER,
D3D11_MESSAGE_ID_REF_OUT_OF_MEMORY,
D3D11_MESSAGE_ID_REF_INFO,
D3D11_MESSAGE_ID_DEVICE_DRAW_VERTEXPOS_OVERFLOW,
D3D11_MESSAGE_ID_DEVICE_DRAWINDEXED_INDEXPOS_OVERFLOW,
D3D11_MESSAGE_ID_DEVICE_DRAWINSTANCED_VERTEXPOS_OVERFLOW,
D3D11_MESSAGE_ID_DEVICE_DRAWINSTANCED_INSTANCEPOS_OVERFLOW,
D3D11_MESSAGE_ID_DEVICE_DRAWINDEXEDINSTANCED_INSTANCEPOS_OVERFLOW,
D3D11_MESSAGE_ID_DEVICE_DRAWINDEXEDINSTANCED_INDEXPOS_OVERFLOW,
D3D11_MESSAGE_ID_DEVICE_DRAW_VERTEX_SHADER_NOT_SET,
D3D11_MESSAGE_ID_DEVICE_SHADER_LINKAGE_SEMANTICNAME_NOT_FOUND,
D3D11_MESSAGE_ID_DEVICE_SHADER_LINKAGE_REGISTERINDEX,
D3D11_MESSAGE_ID_DEVICE_SHADER_LINKAGE_COMPONENTTYPE,
D3D11_MESSAGE_ID_DEVICE_SHADER_LINKAGE_REGISTERMASK,
D3D11_MESSAGE_ID_DEVICE_SHADER_LINKAGE_SYSTEMVALUE,
D3D11_MESSAGE_ID_DEVICE_SHADER_LINKAGE_NEVERWRITTEN_ALWAYSREADS,
D3D11_MESSAGE_ID_DEVICE_DRAW_VERTEX_BUFFER_NOT_SET,
D3D11_MESSAGE_ID_DEVICE_DRAW_INPUTLAYOUT_NOT_SET,
D3D11_MESSAGE_ID_DEVICE_DRAW_CONSTANT_BUFFER_NOT_SET,
```

```
D3D11_MESSAGE_ID_DEVICE_DRAW_CONSTANT_BUFFER_TOO_SMALL,  
D3D11_MESSAGE_ID_DEVICE_DRAW_SAMPLER_NOT_SET,  
D3D11_MESSAGE_ID_DEVICE_DRAW_SHADERRESOURCEVIEW_NOT_SET,  
D3D11_MESSAGE_ID_DEVICE_DRAW_VIEW_DIMENSION_MISMATCH,  
D3D11_MESSAGE_ID_DEVICE_DRAW_VERTEX_BUFFER_STRIDE_TOO_SMALL,  
D3D11_MESSAGE_ID_DEVICE_DRAW_VERTEX_BUFFER_TOO_SMALL,  
D3D11_MESSAGE_ID_DEVICE_DRAW_INDEX_BUFFER_NOT_SET,  
D3D11_MESSAGE_ID_DEVICE_DRAW_INDEX_BUFFER_FORMAT_INVALID,  
D3D11_MESSAGE_ID_DEVICE_DRAW_INDEX_BUFFER_TOO_SMALL,  
D3D11_MESSAGE_ID_DEVICE_DRAW_GS_INPUT_PRIMITIVE_MISMATCH,  
D3D11_MESSAGE_ID_DEVICE_DRAW_RESOURCE_RETURN_TYPE_MISMATCH,  
D3D11_MESSAGE_ID_DEVICE_DRAW_POSITION_NOT_PRESENT,  
D3D11_MESSAGE_ID_DEVICE_DRAW_OUTPUT_STREAM_NOT_SET,  
D3D11_MESSAGE_ID_DEVICE_DRAW_BOUND_RESOURCE_MAPPED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_INVALID_PRIMITIVETOPOLOGY,  
D3D11_MESSAGE_ID_DEVICE_DRAW_VERTEX_OFFSET_UNALIGNED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_VERTEX_STRIDE_UNALIGNED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_INDEX_OFFSET_UNALIGNED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_OUTPUT_STREAM_OFFSET_UNALIGNED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_RESOURCE_FORMAT_LD_UNSUPPORTED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_RESOURCE_FORMAT_SAMPLE_UNSUPPORTED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_RESOURCE_FORMAT_SAMPLE_C_UNSUPPORTED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_RESOURCE_MULTISAMPLE_UNSUPPORTED,  
D3D11_MESSAGE_ID_DEVICE_DRAW_SO_TARGETS_BOUND_WITHOUT_SOURCE,  
D3D11_MESSAGE_ID_DEVICE_DRAW_SO_STRIDE_LARGER_THAN_BUFFER,  
D3D11_MESSAGE_ID_DEVICE_DRAW_OM_RENDER_TARGET_DOES_NOT_SUPPORT_BLENDING,
```

```
D3D11_MESSAGE_ID_DEVICE_DRAW_OM_DUAL_SOURCE_BLENDING_CAN_ONLY_HAVE_RENDER_TARGET_0,  
D3D11_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_AT_FAULT,  
D3D11_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_POSSIBLY_AT_FAULT,  
D3D11_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_NOT_AT_FAULT,  
D3D11_MESSAGE_ID_DEVICE_OPEN_SHARED_RESOURCE_INVALIDARG_RETURN,  
D3D11_MESSAGE_ID_DEVICE_OPEN_SHARED_RESOURCE_OUTOFMEMORY_RETURN,  
D3D11_MESSAGE_ID_DEVICE_OPEN_SHARED_RESOURCE_BADINTERFACE_RETURN,  
D3D11_MESSAGE_ID_DEVICE_DRAW_VIEWPORT_NOT_SET,  
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_TRAILING_DIGIT_IN_SEMANTIC,
```

```
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_TRAILING_DIGIT_IN_SEMANTIC,  
D3D11_MESSAGE_ID_DEVICE_RSSETVIEWPORTS_DENORMFLUSH,  
D3D11_MESSAGE_ID_OMSETRENDERTARGETS_INVALIDVIEW,  
D3D11_MESSAGE_ID_DEVICE_SETTEXTFILTERSIZE_INVALIDDIMENSIONS,  
D3D11_MESSAGE_ID_DEVICE_DRAW_SAMPLER_MISMATCH,  
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_TYPE_MISMATCH,  
D3D11_MESSAGE_ID_BLENDSTATE_GETDESC_LEGACY,  
D3D11_MESSAGE_ID_SHADERRESOURCEVIEW_GETDESC_LEGACY,  
D3D11_MESSAGE_ID_CREATEQUERY_OUTOFMEMORY_RETURN,  
D3D11_MESSAGE_ID_CREATEPREDICATE_OUTOFMEMORY_RETURN,  
D3D11_MESSAGE_ID_CREATECOUNTER_OUTOFRANGE_COUNTER,  
D3D11_MESSAGE_ID_CREATECOUNTER_SIMULTANEOUS_ACTIVE_COUNTERS_EXHAUSTED,  
D3D11_MESSAGE_ID_CREATECOUNTER_UNSUPPORTED_WELLKNOWN_COUNTER,  
D3D11_MESSAGE_ID_CREATECOUNTER_OUTOFMEMORY_RETURN,  
D3D11_MESSAGE_ID_CREATECOUNTER_NONEXCLUSIVE_RETURN,  
D3D11_MESSAGE_ID_CREATECOUNTER_NULLDESC,
```

```
D3D11_MESSAGE_ID_CHECKCOUNTER_OUTOFRANGE_COUNTER,
D3D11_MESSAGE_ID_CHECKCOUNTER_UNSUPPORTED_WELLKNOWN_COUNTER,
D3D11_MESSAGE_ID_SETPREDICATION_INVALID_PREDICATE_STATE,
D3D11_MESSAGE_ID_QUERY_BEGIN_UNSUPPORTED,
D3D11_MESSAGE_ID_PREDICATE_BEGIN_DURING_PREDICATION,
D3D11_MESSAGE_ID_QUERY_BEGIN_DUPLICATE,
D3D11_MESSAGE_ID_QUERY_BEGIN_ABANDONING_PREVIOUS_RESULTS,
D3D11_MESSAGE_ID_PREDICATE_END_DURING_PREDICATION,
D3D11_MESSAGE_ID_QUERY_END_ABANDONING_PREVIOUS_RESULTS,
D3D11_MESSAGE_ID_QUERY_END_WITHOUT_BEGIN,
D3D11_MESSAGE_ID_QUERY_GETDATA_INVALID_DATASIZE,
D3D11_MESSAGE_ID_QUERY_GETDATA_INVALID_FLAGS,
D3D11_MESSAGE_ID_QUERY_GETDATA_INVALID_CALL,
D3D11_MESSAGE_ID_DEVICE_DRAW_PS_OUTPUT_TYPE_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_DRAW_RESOURCE_FORMAT_GATHER_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_DRAW_INVALID_USE_OF_CENTER_MULTISAMPLE_PATTERN,
D3D11_MESSAGE_ID_DEVICE_IASETVERTEXBUFFERS_STRIDE_TOO_LARGE,
D3D11_MESSAGE_ID_DEVICE_IASETVERTEXBUFFERS_INVALIDDRANGE,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_EMPTY_LAYOUT,
D3D11_MESSAGE_ID_DEVICE_DRAW_RESOURCE_SAMPLE_COUNT_MISMATCH,
D3D11_MESSAGE_ID_LIVE_OBJECT_SUMMARY,
D3D11_MESSAGE_ID_LIVE_BUFFER,
D3D11_MESSAGE_ID_LIVE_TEXTURE1D,
D3D11_MESSAGE_ID_LIVE_TEXTURE2D,
D3D11_MESSAGE_ID_LIVE_TEXTURE3D,
D3D11_MESSAGE_ID_LIVE_SHADERRESOURCEVIEW,
D3D11_MESSAGE_ID_LIVE_RENDERTARGETVIEW,
D3D11_MESSAGE_ID_LIVE_DEPTHSTENCILVIEW,
D3D11_MESSAGE_ID_LIVE_VERTEXSHADER,
D3D11_MESSAGE_ID_LIVE_GEOMETRYSHADER,
D3D11_MESSAGE_ID_LIVE_PIXELSHADER,
D3D11_MESSAGE_ID_LIVE_INPUTLAYOUT,
D3D11_MESSAGE_ID_LIVE_SAMPLER,
D3D11_MESSAGE_ID_LIVE_BLENDSTATE,
D3D11_MESSAGE_ID_LIVE_DEPTHSTENCILSTATE,
D3D11_MESSAGE_ID_LIVE_RASTERIZERSTATE,
D3D11_MESSAGE_ID_LIVE_QUERY,
D3D11_MESSAGE_ID_LIVE_PREDICATE,
D3D11_MESSAGE_ID_LIVE_COUNTER,
D3D11_MESSAGE_ID_LIVE_DEVICE,
D3D11_MESSAGE_ID_LIVE_SWAPCHAIN,
D3D11_MESSAGE_ID_D3D10_MESSAGES_END,
D3D11_MESSAGE_ID_D3D10L9_MESSAGES_START = 0x100000,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_STENCIL_NO_TWO_SIDED,
D3D11_MESSAGE_ID_CREATERASTERIZERSTATE_DepthBiasClamp_NOT_SUPPORTED,
D3D11_MESSAGE_ID_CREATESAMPLERSTATE_NO_COMPARISON_SUPPORT,
D3D11_MESSAGE_ID_CREATESAMPLERSTATE_EXCESSIVE_ANISOTROPY,
D3D11_MESSAGE_ID_CREATESAMPLERSTATE_BORDER_OUT_OF_RANGE,
D3D11_MESSAGE_ID_VSSETSAMPLERS_NOT_SUPPORTED,
D3D11_MESSAGE_ID_VSSETSAMPLERS_TOO_MANY_SAMPLERS,
D3D11_MESSAGE_ID_PSSETSAMPLERS_TOO_MANY_SAMPLERS,
D3D11_MESSAGE_ID_CREATEROFILE_NO_ARRAYS,
D3D11_MESSAGE_ID_CREATEROFILE_NO_VB_AND_IB_BIND,
D3D11_MESSAGE_ID_CREATEROFILE_NO_TEXTURE_1D,
D3D11_MESSAGE_ID_CREATEROFILE_DIMENSION_OUT_OF_RANGE,
```

```
D3D11_MESSAGE_ID_CREATERESOURCE_NOT_BINDABLE_AS_SHADER_RESOURCE,
D3D11_MESSAGE_ID_OMSETRENDERTARGETS_TOO_MANY_RENDER_TARGETS,
D3D11_MESSAGE_ID_OMSETRENDERTARGETS_NO_DIFFERING_BIT_DEPTHS,
D3D11_MESSAGE_ID_IASETVERTEXBUFFERS_BAD_BUFFER_INDEX,
D3D11_MESSAGE_ID_DEVICE_RSSETVIEWPORTS_TOO_MANY_VIEWPORTS,
D3D11_MESSAGE_ID_DEVICE_IASETPRIMITIVETOPOLOGY_ADJACENCY_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_RSSETSCISSORRECTS_TOO_MANY_SCISSORS,
D3D11_MESSAGE_ID_COPYRESOURCE_ONLY_TEXTURE_2D_WITHIN_GPU_MEMORY,
D3D11_MESSAGE_ID_COPYRESOURCE_NO_TEXTURE_3D_READBACK,
D3D11_MESSAGE_ID_COPYRESOURCE_NO_TEXTURE_ONLY_READBACK,
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_UNSUPPORTED_FORMAT,
D3D11_MESSAGE_ID_CREATEBLENDSTATE_NO_ALPHA_TO_COVERAGE,
D3D11_MESSAGE_ID_CREATEASTERIZERSTATE_DepthClipEnable_MUST_BE_TRUE,
D3D11_MESSAGE_ID_DRAWINDEXED_STARTINDEXLOCATION_MUST_BE_POSITIVE,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_MUST_USE_LOWEST_LOD,
D3D11_MESSAGE_ID_CREATESAMPLERSTATE_MINLOD_MUST_NOT_BE_FRACTIONAL,
D3D11_MESSAGE_ID_CREATESAMPLERSTATE_MAXLOD_MUST_BE_FLT_MAX,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_FIRSTARRAYSLICE_MUST_BE_ZERO,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_CUBES_MUST_HAVE_6_SIDES,
D3D11_MESSAGE_ID_CREATERESOURCE_NOT_BINDABLE_AS_RENDER_TARGET,
D3D11_MESSAGE_ID_CREATERESOURCE_NO_DWORD_INDEX_BUFFER,
D3D11_MESSAGE_ID_CREATERESOURCE_MSAA_PRECLUDES_SHADER_RESOURCE,
D3D11_MESSAGE_ID_CREATERESOURCE_PRESENTATION_PRECLUDES_SHADER_RESOURCE,
D3D11_MESSAGE_ID_CREATEBLENDSTATE_NO_INDEPENDENT_BLEND_ENABLE,
D3D11_MESSAGE_ID_CREATEBLENDSTATE_NO_INDEPENDENT_WRITE_MASKS,
D3D11_MESSAGE_ID_CREATERESOURCE_NO_STREAM_OUT,
D3D11_MESSAGE_ID_CREATERESOURCE_ONLY_VB_IB_FOR_BUFFERS,
D3D11_MESSAGE_ID_CREATERESOURCE_NO_AUTOGEN_FOR_VOLUMES,
D3D11_MESSAGE_ID_CREATERESOURCE DXGI_FORMAT_R8G8B8A8_CANNOT_BE_SHARED,
D3D11_MESSAGE_ID_VSSHADERRESOURCES_NOT_SUPPORTED,
D3D11_MESSAGE_ID_GEOMETRY_SHADER_NOT_SUPPORTED,
D3D11_MESSAGE_ID_STREAM_OUT_NOT_SUPPORTED,
D3D11_MESSAGE_ID_TEXT_FILTER_NOT_SUPPORTED,
D3D11_MESSAGE_ID_CREATEBLENDSTATE_NO_SEPARATE_ALPHA_BLEND,
D3D11_MESSAGE_ID_CREATEBLENDSTATE_NO_MRT_BLEND,
D3D11_MESSAGE_ID_CREATEBLENDSTATE_OPERATION_NOT_SUPPORTED,
D3D11_MESSAGE_ID_CREATESAMPLERSTATE_NO_MIRRORONCE,
D3D11_MESSAGE_ID_DRAWINSTANCED_NOT_SUPPORTED,
D3D11_MESSAGE_ID_DRAWINDEXEDINSTANCED_NOT_SUPPORTED_BELOW_9_3,
D3D11_MESSAGE_ID_DRAWINDEXED_POINTLIST_UNSUPPORTED,
D3D11_MESSAGE_ID_SETBLENDSTATE_SAMPLE_MASK_CANNOT_BE_ZERO,
```

```
D3D11_MESSAGE_ID_CREATERESOURCE_DIMENSION_EXCEEDS_FEATURE_LEVEL_DEFINITION,
D3D11_MESSAGE_ID_CREATERESOURCE_ONLY_SINGLE_MIP_LEVEL_DEPTH_STENCIL_SUPPORT
D,
D3D11_MESSAGE_ID_DEVICE_RSSETSCISSORRECTS_NEGATIVESCISSOR,
D3D11_MESSAGE_ID_SLOT_ZERO_MUST_BE_D3D10_INPUT_PER_VERTEX_DATA,
D3D11_MESSAGE_ID_CREATERESOURCE_NON_POW_2_MIPMAP,
D3D11_MESSAGE_ID_CREATESAMPLERSTATE_BORDER_NOT_SUPPORTED,
D3D11_MESSAGE_ID_OMSETRENDERTARGETS_NO_SRGB_MRT,
D3D11_MESSAGE_ID_COPYRESOURCE_NO_3D_MISMATCHED_UPDATES,
D3D11_MESSAGE_ID_D3D10L9_MESSAGES_END,
D3D11_MESSAGE_ID_D3D11_MESSAGES_START = 0x200000,
D3D11_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDFLAGS,
```

```
D3D11_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDCLASSLINKAGE,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDCLASSLINKAGE,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMSTREAMS,

D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTREAMTORASTERIZER,
    D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDSTREAMS,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDCLASSLINKAGE,
D3D11_MESSAGE_ID_CREATEPIXELSHADER_INVALIDCLASSLINKAGE,
D3D11_MESSAGE_ID_CREATEDFERREDCONTEXT_INVALID_COMMANDLISTFLAGS,
D3D11_MESSAGE_ID_CREATEDFERREDCONTEXT_SINGLETHREADED,
D3D11_MESSAGE_ID_CREATEDFERREDCONTEXT_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATEDFERREDCONTEXT_INVALID_CALL_RETURN,
D3D11_MESSAGE_ID_CREATEDFERREDCONTEXT_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_FINISHDISPLAYLIST_ONIMMEDIATECONTEXT,
D3D11_MESSAGE_ID_FINISHDISPLAYLIST_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_FINISHDISPLAYLIST_INVALID_CALL_RETURN,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTREAM,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDENTRIES,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDSTRIDES,
D3D11_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMSTRIDES,
D3D11_MESSAGE_ID_DEVICE_HSSETSHADERRESOURCES_HAZARD,
D3D11_MESSAGE_ID_DEVICE_HSSETCONSTANTBUFFERS_HAZARD,
D3D11_MESSAGE_ID_HSSETSHADERRESOURCES_UNBINDELETEINGOBJECT,
D3D11_MESSAGE_ID_HSSETCONSTANTBUFFERS_UNBINDELETEINGOBJECT,
D3D11_MESSAGE_ID_CREATEHULLSHADER_INVALIDCALL,
D3D11_MESSAGE_ID_CREATEHULLSHADER_OUTOFMEMORY,
D3D11_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADERBYTECODE,
D3D11_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADERTYPE,
D3D11_MESSAGE_ID_CREATEHULLSHADER_INVALIDCLASSLINKAGE,
D3D11_MESSAGE_ID_DEVICE_HSSETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_HSSETCONSTANTBUFFERS_INVALIDBUFFER,
D3D11_MESSAGE_ID_DEVICE_HSSETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_HSSETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_HSGETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_HSGETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_HSGETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_DSSETSHADERRESOURCES_HAZARD,
D3D11_MESSAGE_ID_DEVICE_DSSETCONSTANTBUFFERS_HAZARD,
D3D11_MESSAGE_ID_DSSETSHADERRESOURCES_UNBINDELETEINGOBJECT,
D3D11_MESSAGE_ID_DSSETCONSTANTBUFFERS_UNBINDELETEINGOBJECT,
D3D11_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDCALL,
D3D11_MESSAGE_ID_CREATEDOMAINSHADER_OUTOFMEMORY,
D3D11_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERBYTECODE,
D3D11_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERTYPE,
D3D11_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDCLASSLINKAGE,
D3D11_MESSAGE_ID_DEVICE_DSSETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_DSSETCONSTANTBUFFERS_INVALIDBUFFER,
D3D11_MESSAGE_ID_DEVICE_DSSETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_DSSETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_DSGETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_DSGETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_DSGETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_DRAW_HS_XOR_DS_MISMATCH,
D3D11_MESSAGE_ID_DEFERRED_CONTEXT_REMOVAL_PROCESS_AT_FAULT,
```

```
D3D11_MESSAGE_ID_DEVICE_DRAWINDIRECT_INVALID_ARG_BUFFER,
D3D11_MESSAGE_ID_DEVICE_DRAWINDIRECT_OFFSET_UNALIGNED,
D3D11_MESSAGE_ID_DEVICE_DRAWINDIRECT_OFFSET_OVERFLOW,
D3D11_MESSAGE_ID_RESOURCE_MAP_INVALIDMAPTYPE,
D3D11_MESSAGE_ID_RESOURCE_MAP_INVALIDSUBRESOURCE,
D3D11_MESSAGE_ID_RESOURCE_MAP_INVALIDFLAGS,
D3D11_MESSAGE_ID_RESOURCE_MAP_ALREADYMAPPED,
D3D11_MESSAGE_ID_RESOURCE_MAP_DEVICEREMOVED_RETURN,
D3D11_MESSAGE_ID_RESOURCE_MAP_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_RESOURCE_MAP_WITHOUT_INITIAL_DISCARD,
D3D11_MESSAGE_ID_RESOURCE_UNMAP_INVALIDSUBRESOURCE,
D3D11_MESSAGE_ID_RESOURCE_UNMAP_NOTMAPPED,
D3D11_MESSAGE_ID_DEVICE_DRAW_RASTERIZING_CONTROL_POINTS,
D3D11_MESSAGE_ID_DEVICE_IASETPRIMITIVETOPOLOGY_TOPOLOGY_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_DRAW_HS_DS_SIGNATURE_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_DRAW_HULL_SHADER_INPUT_TOPOLOGY_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_DRAW_HS_DS_CONTROL_POINT_COUNT_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_DRAW_HS_DS_TESSELLATOR_DOMAIN_MISMATCH,
D3D11_MESSAGE_ID_CREATE_CONTEXT,
D3D11_MESSAGE_ID_LIVE_CONTEXT,
D3D11_MESSAGE_ID_DESTROY_CONTEXT,
D3D11_MESSAGE_ID_CREATE_BUFFER,
D3D11_MESSAGE_ID_LIVE_BUFFER_WIN7,
D3D11_MESSAGE_ID_DESTROY_BUFFER,
D3D11_MESSAGE_ID_CREATE_TEXTURE1D,
D3D11_MESSAGE_ID_LIVE_TEXTURE1D_WIN7,
D3D11_MESSAGE_ID_DESTROY_TEXTURE1D,
D3D11_MESSAGE_ID_CREATE_TEXTURE2D,
D3D11_MESSAGE_ID_LIVE_TEXTURE2D_WIN7,
D3D11_MESSAGE_ID_DESTROY_TEXTURE2D,
D3D11_MESSAGE_ID_CREATE_TEXTURE3D,
D3D11_MESSAGE_ID_LIVE_TEXTURE3D_WIN7,
D3D11_MESSAGE_ID_DESTROY_TEXTURE3D,
D3D11_MESSAGE_ID_CREATE_SHADERRESOURCEVIEW,
D3D11_MESSAGE_ID_LIVE_SHADERRESOURCEVIEW_WIN7,
D3D11_MESSAGE_ID_DESTROY_SHADERRESOURCEVIEW,
D3D11_MESSAGE_ID_CREATE_RENDERTARGETVIEW,
D3D11_MESSAGE_ID_LIVE_RENDERTARGETVIEW_WIN7,
D3D11_MESSAGE_ID_DESTROY_RENDERTARGETVIEW,
D3D11_MESSAGE_ID_CREATE_DEPTHSTENCILVIEW,
D3D11_MESSAGE_ID_LIVE_DEPTHSTENCILVIEW_WIN7,
D3D11_MESSAGE_ID_DESTROY_DEPTHSTENCILVIEW,
D3D11_MESSAGE_ID_CREATE_VERTEXSHADER,
D3D11_MESSAGE_ID_LIVE_VERTEXSHADER_WIN7,
D3D11_MESSAGE_ID_DESTROY_VERTEXSHADER,
D3D11_MESSAGE_ID_CREATE_HULLSHADER,
D3D11_MESSAGE_ID_LIVE_HULLSHADER,
D3D11_MESSAGE_ID_DESTROY_HULLSHADER,
D3D11_MESSAGE_ID_CREATE_DOMAINSHADER,
D3D11_MESSAGE_ID_LIVE_DOMAINSHADER,
D3D11_MESSAGE_ID_DESTROY_DOMAINSHADER,
D3D11_MESSAGE_ID_CREATE_GEOMETRYSHADER,
D3D11_MESSAGE_ID_LIVE_GEOMETRYSHADER_WIN7,
D3D11_MESSAGE_ID_DESTROY_GEOMETRYSHADER,
D3D11_MESSAGE_ID_CREATE_PIXELSHADER,
```

```
D3D11_MESSAGE_ID_LIVE_PIXELSHADER_WIN7,
D3D11_MESSAGE_ID_DESTROY_PIXELSHADER,
D3D11_MESSAGE_ID_CREATE_INPUTLAYOUT,
D3D11_MESSAGE_ID_LIVE_INPUTLAYOUT_WIN7,
D3D11_MESSAGE_ID_DESTROY_INPUTLAYOUT,
D3D11_MESSAGE_ID_CREATE_SAMPLER,
D3D11_MESSAGE_ID_LIVE_SAMPLER_WIN7,
D3D11_MESSAGE_ID_DESTROY_SAMPLER,
D3D11_MESSAGE_ID_CREATE_BLENDSTATE,
D3D11_MESSAGE_ID_LIVE_BLENDSTATE_WIN7,
D3D11_MESSAGE_ID_DESTROY_BLENDSTATE,
D3D11_MESSAGE_ID_CREATE_DEPTHSTENCILSTATE,
D3D11_MESSAGE_ID_LIVE_DEPTHSTENCILSTATE_WIN7,
D3D11_MESSAGE_ID_DESTROY_DEPTHSTENCILSTATE,
D3D11_MESSAGE_ID_CREATE_RASTERIZERSTATE,
D3D11_MESSAGE_ID_LIVE_RASTERIZERSTATE_WIN7,
D3D11_MESSAGE_ID_DESTROY_RASTERIZERSTATE,
D3D11_MESSAGE_ID_CREATE_QUERY,
D3D11_MESSAGE_ID_LIVE_QUERY_WIN7,
D3D11_MESSAGE_ID_DESTROY_QUERY,
D3D11_MESSAGE_ID_CREATE_PREDICATE,
D3D11_MESSAGE_ID_LIVE_PREDICATE_WIN7,
D3D11_MESSAGE_ID_DESTROY_PREDICATE,
D3D11_MESSAGE_ID_CREATE_COUNTER,
D3D11_MESSAGE_ID_DESTROY_COUNTER,
D3D11_MESSAGE_ID_CREATE_COMMANDLIST,
D3D11_MESSAGE_ID_LIVE_COMMANDLIST,
D3D11_MESSAGE_ID_DESTROY_COMMANDLIST,
D3D11_MESSAGE_ID_CREATE_CLASSINSTANCE,
D3D11_MESSAGE_ID_LIVE_CLASSINSTANCE,
D3D11_MESSAGE_ID_DESTROY_CLASSINSTANCE,
D3D11_MESSAGE_ID_CREATE_CLASSLINKAGE,
D3D11_MESSAGE_ID_LIVE_CLASSLINKAGE,
D3D11_MESSAGE_ID_DESTROY_CLASSLINKAGE,
D3D11_MESSAGE_ID_LIVE_DEVICE_WIN7,
D3D11_MESSAGE_ID_LIVE_OBJECT_SUMMARY_WIN7,
D3D11_MESSAGE_ID_CREATE_COMPUTESHADER,
D3D11_MESSAGE_ID_LIVE_COMPUTESHADER,
D3D11_MESSAGE_ID_DESTROY_COMPUTESHADER,
D3D11_MESSAGE_ID_CREATE_UNORDEREDACCESSVIEW,
D3D11_MESSAGE_ID_LIVE_UNORDEREDACCESSVIEW,
D3D11_MESSAGE_ID_DESTROY_UNORDEREDACCESSVIEW,
D3D11_MESSAGE_ID_DEVICE_SETSHADER_INTERFACES_FEATURELEVEL,
D3D11_MESSAGE_ID_DEVICE_SETSHADER_INTERFACE_COUNT_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_SETSHADER_INVALID_INSTANCE,
D3D11_MESSAGE_ID_DEVICE_SETSHADER_INVALID_INSTANCE_INDEX,
D3D11_MESSAGE_ID_DEVICE_SETSHADER_INVALID_INSTANCE_TYPE,
D3D11_MESSAGE_ID_DEVICE_SETSHADER_INVALID_INSTANCE_DATA,
D3D11_MESSAGE_ID_DEVICE_SETSHADER_UNBOUND_INSTANCE_DATA,
D3D11_MESSAGE_ID_DEVICE_SETSHADER_INSTANCE_DATA_BINDINGS,
D3D11_MESSAGE_ID_DEVICE_CREATESHADER_CLASSLINKAGE_FULL,
D3D11_MESSAGE_ID_DEVICE_CHECKFEATURESUPPORT_UNRECOGNIZED_FEATURE,
D3D11_MESSAGE_ID_DEVICE_CHECKFEATURESUPPORT_MISMATCHED_DATA_SIZE,
D3D11_MESSAGE_ID_DEVICE_CHECKFEATURESUPPORT_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_DEVICE_CSSETSHADERRESOURCES_HAZARD,
```

```
D3D11_MESSAGE_ID_DEVICE_CSSETCONSTANTBUFFERS_HAZARD,
D3D11_MESSAGE_ID_CSSETSHADERRESOURCES_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_CSSETCONSTANTBUFFERS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDCALL,
D3D11_MESSAGE_ID_CREATECOMPUTESHADER_OUTOFMEMORY,
D3D11_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDSHADERBYTECODE,
D3D11_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDSHADERTYPE,
D3D11_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDCLASSLINKAGE,
D3D11_MESSAGE_ID_DEVICE_CSSETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_CSSETCONSTANTBUFFERS_INVALIDBUFFER,
D3D11_MESSAGE_ID_DEVICE_CSSETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_CSSETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_CSGETSHADERRESOURCES_VIEWS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_CSGETCONSTANTBUFFERS_BUFFERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_CSGETSAMPLERS_SAMPLERS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_CREATEHULLSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEFLOATOPSNOTSUPPORTED,

D3D11_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEFLOATOPSNOTSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDSTRUCTURESTRIDE,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDFLAGS,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDRESOURCE,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDESC,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFORMAT,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDIMENSIONS,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_UNRECOGNIZEDFORMAT,
D3D11_MESSAGE_ID_DEVICE_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS_HAZARD,

D3D11_MESSAGE_ID_DEVICE_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS_OVERLAPPING_OLD_SLOTS,
D3D11_MESSAGE_ID_DEVICE_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS_NO_OP,
D3D11_MESSAGE_ID_CSSETUNORDEREDACCESSVIEWS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_PSSETUNORDEREDACCESSVIEWS_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_TOOMANYOBJECTS,
D3D11_MESSAGE_ID_DEVICE_CSSETUNORDEREDACCESSVIEWS_HAZARD,
D3D11_MESSAGE_ID_CLEARUNORDEREDACCESSVIEW_DENORMFLUSH,
D3D11_MESSAGE_ID_DEVICE_CSSETUNORDEREDACCESSS_VIEWS_EMPTY,
D3D11_MESSAGE_ID_DEVICE_CSGETUNORDEREDACCESSS_VIEWS_EMPTY,
D3D11_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFLAGS,
D3D11_MESSAGE_ID_CREATESHADERRESOURCEVIEW_TOOMANYOBJECTS,
D3D11_MESSAGE_ID_DEVICE_DISPATCHINDIRECT_INVALID_ARG_BUFFER,
D3D11_MESSAGE_ID_DEVICE_DISPATCHINDIRECT_OFFSET_UNALIGNED,
D3D11_MESSAGE_ID_DEVICE_DISPATCHINDIRECT_OFFSET_OVERFLOW,
D3D11_MESSAGE_ID_DEVICE_SETRESOURCEMINLOD_INVALIDCONTEXT,
D3D11_MESSAGE_ID_DEVICE_SETRESOURCEMINLOD_INVALIDRESOURCE,
D3D11_MESSAGE_ID_DEVICE_SETRESOURCEMINLOD_INVALIDMINLOD,
D3D11_MESSAGE_ID_DEVICE_GETRESOURCEMINLOD_INVALIDCONTEXT,
D3D11_MESSAGE_ID_DEVICE_GETRESOURCEMINLOD_INVALIDRESOURCE,
```

```
D3D11_MESSAGE_ID_OMSETDEPTHSTENCIL_UNBINDELETEINGOBJECT,
D3D11_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_DEPTH_READONLY,
D3D11_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_STENCIL_READONLY,
D3D11_MESSAGE_ID_CHECKFEATURESUPPORT_FORMAT_DEPRECATED,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_RETURN_TYPE_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_NOT_SET,
D3D11_MESSAGE_ID_DEVICE_DRAW_UNORDEREDACCESSVIEW_RENDERTARGETVIEW_OVERLAP,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_DIMENSION_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_APPEND_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_ATOMICS_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_STRUCTURE_STRIDE_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_BUFFER_TYPE_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_RAW_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_FORMAT_LD_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_FORMAT_STORE_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_ATOMIC_ADD_UNSUPPORTED,

D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_ATOMIC_BITWISE_OPS_UNSUPPORTED,

D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_ATOMIC_CMPSTORE_CMPEXCHANGE_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_ATOMIC_EXCHANGE_UNSUPPORTED,

D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_ATOMIC_SIGNED_MINMAX_UNSUPPORTED
,

D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_ATOMIC_UNSIGNED_MINMAX_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_DISPATCH_BOUND_RESOURCE_MAPPED,
D3D11_MESSAGE_ID_DEVICE_DISPATCH_THREADGROUPCOUNT_OVERFLOW,
D3D11_MESSAGE_ID_DEVICE_DISPATCH_THREADGROUPCOUNT_ZERO,
D3D11_MESSAGE_ID_DEVICE_SHADERRESOURCEVIEW_STRUCTURE_STRIDE_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_SHADERRESOURCEVIEW_BUFFER_TYPE_MISMATCH,
D3D11_MESSAGE_ID_DEVICE_SHADERRESOURCEVIEW_RAW_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_DISPATCH_UNSUPPORTED,
D3D11_MESSAGE_ID_DEVICE_DISPATCHINDIRECT_UNSUPPORTED,
D3D11_MESSAGE_ID_COPYSTRUCTURECOUNT_INVALIDOFFSET,
D3D11_MESSAGE_ID_COPYSTRUCTURECOUNT_LARGEOFFSET,
D3D11_MESSAGE_ID_COPYSTRUCTURECOUNT_INVALIDDESTINATIONSTATE,
D3D11_MESSAGE_ID_COPYSTRUCTURECOUNT_INVALIDSOURCESTATE,
D3D11_MESSAGE_ID_CHECKFORMATSUPPORT_FORMAT_NOT_SUPPORTED,
D3D11_MESSAGE_ID_DEVICE_CSSETUNORDEREDACCESSVIEWS_INVALIDVIEW,
D3D11_MESSAGE_ID_DEVICE_CSSETUNORDEREDACCESSVIEWS_INVALIDOFFSET,
D3D11_MESSAGE_ID_DEVICE_CSSETUNORDEREDACCESSVIEWS_TOOMANYVIEWS,
D3D11_MESSAGE_ID_CLEARUNORDEREDACCESSVIEWFLOAT_INVALIDFORMAT,
D3D11_MESSAGE_ID_DEVICE_UNORDEREDACCESSVIEW_COUNTER_UNSUPPORTED,
D3D11_MESSAGE_ID_REF_WARNING,
D3D11_MESSAGE_ID_DEVICE_DRAW_PIXEL_SHADER_WITHOUT_RTV_OR_DSV,
D3D11_MESSAGE_ID_SHADER_ABORT,
D3D11_MESSAGE_ID_SHADER_MESSAGE,
D3D11_MESSAGE_ID_SHADER_ERROR,
D3D11_MESSAGE_ID_OFFERRESOURCES_INVALIDRESOURCE,
D3D11_MESSAGE_ID_HSSETSAMPLERS_UNBINDELETEINGOBJECT,
D3D11_MESSAGE_ID_DSSETSAMPLERS_UNBINDELETEINGOBJECT,
D3D11_MESSAGE_ID_CSSETSAMPLERS_UNBINDELETEINGOBJECT,
```

```
D3D11_MESSAGE_ID_HSSETSHADER_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_DSSETSHADER_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_CSSETSHADER_UNBINDDLETINGOBJECT,
D3D11_MESSAGE_ID_ENQUEUESETEVENT_INVALIDARG_RETURN,
D3D11_MESSAGE_ID_ENQUEUESETEVENT_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_ENQUEUESETEVENT_ACCESSDENIED_RETURN,

D3D11_MESSAGE_ID_DEVICE_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS_NUMUAVS_IN
VALIDRANGE,
D3D11_MESSAGE_ID_USE_OF_ZERO_REFCOUNT_OBJECT,
D3D11_MESSAGE_ID_D3D11_MESSAGES_END,
D3D11_MESSAGE_ID_D3D11_1_MESSAGES_START = 0x300000,
D3D11_MESSAGE_ID_CREATE_VIDEODECODER,
D3D11_MESSAGE_ID_CREATE_VIDEOPROCESSORENUM,
D3D11_MESSAGE_ID_CREATE_VIDEOPROCESSOR,
D3D11_MESSAGE_ID_CREATE_DECODEROUTPUTVIEW,
D3D11_MESSAGE_ID_CREATE_PROCESSORINPUTVIEW,
D3D11_MESSAGE_ID_CREATE_PROCESSOROUTPUTVIEW,
D3D11_MESSAGE_ID_CREATE_DEVICECONTEXTSTATE,
D3D11_MESSAGE_ID_LIVE_VIDEODECODER,
D3D11_MESSAGE_ID_LIVE_VIDEOPROCESSORENUM,
D3D11_MESSAGE_ID_LIVE_VIDEOPROCESSOR,
D3D11_MESSAGE_ID_LIVE_DECODEROUTPUTVIEW,
D3D11_MESSAGE_ID_LIVE_PROCESSORINPUTVIEW,
D3D11_MESSAGE_ID_LIVE_PROCESSOROUTPUTVIEW,
D3D11_MESSAGE_ID_LIVE_DEVICECONTEXTSTATE,
D3D11_MESSAGE_ID_DESTROY_VIDEODECODER,
D3D11_MESSAGE_ID_DESTROY_VIDEOPROCESSORENUM,
D3D11_MESSAGE_ID_DESTROY_VIDEOPROCESSOR,
D3D11_MESSAGE_ID_DESTROY_DECODEROUTPUTVIEW,
D3D11_MESSAGE_ID_DESTROY_PROCESSORINPUTVIEW,
D3D11_MESSAGE_ID_DESTROY_PROCESSOROUTPUTVIEW,
D3D11_MESSAGE_ID_DESTROY_DEVICECONTEXTSTATE,
D3D11_MESSAGE_ID_CREATEDeviceCONTEXTSTATE_INVALIDFLAGS,
D3D11_MESSAGE_ID_CREATEDeviceCONTEXTSTATE_INVALIDFEATURELEVEL,
D3D11_MESSAGE_ID_CREATEDeviceCONTEXTSTATE_FEATURELEVELS_NOT_SUPPORTED,
D3D11_MESSAGE_ID_CREATEDeviceCONTEXTSTATE_INVALIDREFIID,
D3D11_MESSAGE_ID_DEVICE_DISCARDVIEW_INVALIDVIEW,
D3D11_MESSAGE_ID_COPYSUBRESOURCEREGION1_INVALIDCOPYFLAGS,
D3D11_MESSAGE_ID_UPDATESUBRESOURCE1_INVALIDCOPYFLAGS,
D3D11_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDFORCEDSAMPLECOUNT,
D3D11_MESSAGE_ID_CREATEVIDEODECODER_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATEVIDEODECODER_NULLPARAM,
D3D11_MESSAGE_ID_CREATEVIDEODECODER_INVALIDFORMAT,
D3D11_MESSAGE_ID_CREATEVIDEODECODER_ZEROWIDTHHEIGHT,
D3D11_MESSAGE_ID_CREATEVIDEODECODER_DRIVER_INVALIDBUFFERSIZE,
D3D11_MESSAGE_ID_CREATEVIDEODECODER_DRIVER_INVALIDBUFFERUSAGE,
D3D11_MESSAGE_ID_GETVIDEODECODERPROFILECOUNT_OUTOFMEMORY,
D3D11_MESSAGE_ID_GETVIDEODECODERPROFILE_NULLPARAM,
D3D11_MESSAGE_ID_GETVIDEODECODERPROFILE_INVALIDINDEX,
D3D11_MESSAGE_ID_GETVIDEODECODERPROFILE_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CHECKVIDEODECODERFORMAT_NULLPARAM,
D3D11_MESSAGE_ID_CHECKVIDEODECODERFORMAT_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_GETVIDEODECODERCONFIGCOUNT_NULLPARAM,
D3D11_MESSAGE_ID_GETVIDEODECODERCONFIGCOUNT_OUTOFMEMORY_RETURN,
```

```
D3D11_MESSAGE_ID_GETVIDEODECODERCONFIG_NULLPARAM,
D3D11_MESSAGE_ID_GETVIDEODECODERCONFIG_INVALIDINDEX,
D3D11_MESSAGE_ID_GETVIDEODECODERCONFIG_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_GETDECODERCREATIONPARAMS_NULLPARAM,
D3D11_MESSAGE_ID_GETDECODERDRIVERHANDLE_NULLPARAM,
D3D11_MESSAGE_ID_GETDECODERBUFFER_NULLPARAM,
D3D11_MESSAGE_ID_GETDECODERBUFFER_INVALIDDBUFFER,
D3D11_MESSAGE_ID_GETDECODERBUFFER_INVALIDTYPE,
D3D11_MESSAGE_ID_GETDECODERBUFFER_LOCKED,
D3D11_MESSAGE_ID_RELEASEDECODERBUFFER_NULLPARAM,
D3D11_MESSAGE_ID_RELEASEDECODERBUFFER_INVALIDTYPE,
D3D11_MESSAGE_ID_RELEASEDECODERBUFFER_NOTLOCKED,
D3D11_MESSAGE_ID_DECODERBEGINFRAME_NULLPARAM,
D3D11_MESSAGE_ID_DECODERBEGINFRAME_HAZARD,
D3D11_MESSAGE_ID_DECODERENDFRAME_NULLPARAM,
D3D11_MESSAGE_ID_SUBMITDECODERBUFFERS_NULLPARAM,
D3D11_MESSAGE_ID_SUBMITDECODERBUFFERS_INVALIDTYPE,
D3D11_MESSAGE_ID_DECODEREXTENSION_NULLPARAM,
D3D11_MESSAGE_ID_DECODEREXTENSION_INVALIDRESOURCE,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSORENUMERATOR_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSORENUMERATOR_NULLPARAM,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSORENUMERATOR_INVALIDFRAMEFORMAT,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSORENUMERATOR_INVALIDUSAGE,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSORENUMERATOR_INVALIDINPUTFRAMERATE,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSORENUMERATOR_INVALIDOUTPUTFRAMERATE,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSORENUMERATOR_INVALIDWIDTHHEIGHT,
D3D11_MESSAGE_ID_GETVIDEOPROCESSORCONTENTDESC_NULLPARAM,
D3D11_MESSAGE_ID_CHECKVIDEOPROCESSORFORMAT_NULLPARAM,
D3D11_MESSAGE_ID_GETVIDEOPROCESSORCAPS_NULLPARAM,
D3D11_MESSAGE_ID_GETVIDEOPROCESSORRATECONVERSIONCAPS_NULLPARAM,
D3D11_MESSAGE_ID_GETVIDEOPROCESSORRATECONVERSIONCAPS_INVALIDINDEX,
D3D11_MESSAGE_ID_GETVIDEOPROCESSORCUSTOMRATE_NULLPARAM,
D3D11_MESSAGE_ID_GETVIDEOPROCESSORCUSTOMRATE_INVALIDINDEX,
D3D11_MESSAGE_ID_GETVIDEOPROCESSORFILTERRANGE_NULLPARAM,
D3D11_MESSAGE_ID_GETVIDEOPROCESSORFILTERRANGE_UNSUPPORTED,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSOR_OUTOFMEMORY_RETURN,
D3D11_MESSAGE_ID_CREATEVIDEOPROCESSOR_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTTARGETRECT_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTBACKGROUNDCOLOR_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTBACKGROUNDCOLOR_INVALIDALPHA,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTCOLORSPACE_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTALPHAFILLMODE_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTALPHAFILLMODE_UNSUPPORTED,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTALPHAFILLMODE_INVALIDSTREAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTALPHAFILLMODE_INVALIDFILLMODE,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTCONSTRICKTION_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTSTEREOMODE_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTSTEREOMODE_UNSUPPORTED,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTTEXTENSION_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORGETOUTPUTTARGETRECT_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORGETOUTPUTBACKGROUNDCOLOR_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORGETOUTPUTCOLORSPACE_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORGETOUTPUTALPHAFILLMODE_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORGETOUTPUTCONSTRICKTION_NULLPARAM,
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTCONSTRICKTION_UNSUPPORTED,
```

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTCONSTRICKTION\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTSTEREOMODE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTEXTENSION\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFRAMEFORMAT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFRAMEFORMAT\_INVALIDFORMAT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFRAMEFORMAT\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMCOLORSPACE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMCOLORSPACE\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMOUTPUTRATE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMOUTPUTRATE\_INVALIDRATE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMOUTPUTRATE\_INVALIDFLAG,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMOUTPUTRATE\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSOURCERECT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSOURCERECT\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSOURCERECT\_INVALIDRECT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMDESTRECT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMDESTRECT\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMDESTRECT\_INVALIDRECT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMALPHA\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMALPHA\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMALPHA\_INVALIDALPHA,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPALETTE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPALETTE\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPALETTE\_INVALIDCOUNT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPALETTE\_INVALIDALPHA,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPIXELASPECTRATIO\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPIXELASPECTRATIO\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPIXELASPECTRATIO\_INVALIDRATIO,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMLUMAKEY\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMLUMAKEY\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMLUMAKEY\_INVALIDRANGE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMLUMAKEY\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_FLIPUNSUPPORTED,  
  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_MONOOFFSETUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_FORMATUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_INVALIDFORMAT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMAUTOPROCESSINGMODE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMAUTOPROCESSINGMODE\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_INVALIDFILTER,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_INVALIDLEVEL,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMEXTENSION\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMEXTENSION\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMFRAMEFORMAT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMCOLORSPACE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMOUTPUTRATE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMSOURCERECT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMDESTRECT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMALPHA\_NULLPARAM,

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMPALETTE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMPIXELASPECTRATIO\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMLUMAKEY\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMSTEREOFORMAT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMAUTOPROCESSINGMODE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMFILTER\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMEXTENSION\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMEXTENSION\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDSTREAMCOUNT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_TARGETRECT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDOUTPUT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDPASTFRAMES,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDFUTUREFRAMES,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDSOURCERECT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDDESTRECT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDINPUTRESOURCE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDARRAYSIZE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDARRAY,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_RIGHTEXPECTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_RIGHTNOTEXPECTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_STEREOENABLED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDRIGHTRESOURCE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_NOSTEREOSTREAMS,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INPUTHAZARD,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_OUTPUTHAZARD,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_OUTOFMEMORY\_RETURN,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_INVALIDTYPE,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_INVALIDBIND,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_UNSUPPORTEDFORMAT,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_INVALIDMIP,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_UNSUPPORTEMIP,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_INVALIDARRAYSIZE,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_INVALIDARRAY,  
D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROOUTPUTVIEW\_INVALIDDIMENSION,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_OUTOFMEMORY\_RETURN,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDTYPE,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDBIND,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDMISC,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDUSAGE,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDFORMAT,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDFOURCC,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDMIP,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_UNSUPPORTEDMIP,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDARRAYSIZE,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDARRAY,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDDIMENSION,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_OUTOFMEMORY\_RETURN,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDTYPE,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDBIND,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDFORMAT,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDMIP,

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_UNSUPPORTEDMIP,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_UNSUPPORTEDARRAY,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDARRAY,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDDIMENSION,  
D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INVALID\_USE\_OF\_FORCED\_SAMPLE\_COUNT,  
D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDLOGICOPS,  
D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDDARRAYWITHDECODER,  
D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDDARRAYWITHDECODER,  
D3D11\_MESSAGE\_ID\_CREATERENDERTARGETVIEW\_INVALIDDARRAYWITHDECODER,  
D3D11\_MESSAGE\_ID\_DEVICE\_LOCKEDOUT\_INTERFACE,  
D3D11\_MESSAGE\_ID\_REF\_WARNING\_ATOMIC\_INCONSISTENT,  
D3D11\_MESSAGE\_ID\_REF\_WARNING\_READING\_UNINITIALIZED\_RESOURCE,  
D3D11\_MESSAGE\_ID\_REF\_WARNING\_RAW\_HAZARD,  
D3D11\_MESSAGE\_ID\_REF\_WARNING\_WAR\_HAZARD,  
D3D11\_MESSAGE\_ID\_REF\_WARNING\_WAW\_HAZARD,  
D3D11\_MESSAGE\_ID\_CREATECRYPTOSESSION\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CREATECRYPTOSESSION\_OUTOFMEMORY\_RETURN,  
D3D11\_MESSAGE\_ID\_GETCRYPTOTYPE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETDECODERPROFILE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONCERTIFICATESIZE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONCERTIFICATE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONCERTIFICATE\_WRONGSIZE,  
D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONHANDLE\_WRONGSIZE,  
D3D11\_MESSAGE\_ID\_NEGOTIATECRYPTOSESSIONKEYEXCHANGE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_WRONGDEVICE,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_DST\_WRONGDEVICE,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_FORMAT\_MISMATCH,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SIZE\_MISMATCH,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_MULTISAMPLED,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_DST\_NOT\_STAGING,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_MAPPED,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_DST\_MAPPED,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_OFFERED,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_DST\_OFFERED,  
D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_CONTENT\_UNDEFINED,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_WRONGDEVICE,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_WRONGDEVICE,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_FORMAT\_MISMATCH,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SIZE\_MISMATCH,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_MULTISAMPLED,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_NOT\_STAGING,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_NOT\_RENDER\_TARGET,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_MAPPED,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_MAPPED,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_OFFERED,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_OFFERED,  
D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_CONTENT\_UNDEFINED,  
D3D11\_MESSAGE\_ID\_STARTSESSIONKEYREFRESH\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_STARTSESSIONKEYREFRESH\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_FINISHSESSIONKEYREFRESH\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETENCRYPTIONBLTKEY\_NULLPARAM,

D3D11\_MESSAGE\_ID\_GETENCRYPTIONBLTKEY\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_GETCONTENTPROTECTIONCAPS\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CHECKCRYPTOKEYEXCHANGE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CHECKCRYPTOKEYEXCHANGE\_INVALIDINDEX,  
D3D11\_MESSAGE\_ID\_CREATEAUTENTICATEDCHANNEL\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CREATEAUTENTICATEDCHANNEL\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_CREATEAUTENTICATEDCHANNEL\_INVALIDTYPE,  
D3D11\_MESSAGE\_ID\_CREATEAUTENTICATEDCHANNEL\_OUTOFMEMORY\_RETURN,  
D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATESIZE\_INVALIDCHANNEL,  
D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATESIZE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATE\_INVALIDCHANNEL,  
D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATE\_WRONGSIZE,  
D3D11\_MESSAGE\_ID\_NEGOTIAEAUTHENTICATEDCHANNELKEYEXCHANGE\_INVALIDCHANNEL,  
D3D11\_MESSAGE\_ID\_NEGOTIAEAUTHENTICATEDCHANNELKEYEXCHANGE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_QUERYAUTENTICATEDCHANNEL\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_QUERYAUTENTICATEDCHANNEL\_WRONGCHANNEL,  
D3D11\_MESSAGE\_ID\_QUERYAUTENTICATEDCHANNEL\_UNSUPPORTEDQUERY,  
D3D11\_MESSAGE\_ID\_QUERYAUTENTICATEDCHANNEL\_WRONGSIZE,  
D3D11\_MESSAGE\_ID\_QUERYAUTENTICATEDCHANNEL\_INVALIDPROCESSINDEX,  
D3D11\_MESSAGE\_ID\_CONFIGUREAUTENTICATEDCHANNEL\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CONFIGUREAUTENTICATEDCHANNEL\_WRONGCHANNEL,  
D3D11\_MESSAGE\_ID\_CONFIGUREAUTENTICATEDCHANNEL\_UNSUPPORTEDCONFIGURE,  
D3D11\_MESSAGE\_ID\_CONFIGUREAUTENTICATEDCHANNEL\_WRONGSIZE,  
D3D11\_MESSAGE\_ID\_CONFIGUREAUTENTICATEDCHANNEL\_INVALIDPROCESSIDTYPE,  
D3D11\_MESSAGE\_ID\_VSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT,  
D3D11\_MESSAGE\_ID\_DSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT,  
D3D11\_MESSAGE\_ID\_HSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT,  
D3D11\_MESSAGE\_ID\_GSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT,  
D3D11\_MESSAGE\_ID\_PSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT,  
D3D11\_MESSAGE\_ID\_CSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT,  
D3D11\_MESSAGE\_ID\_NEGOTIATECRYPTOSESSIONKEYEXCHANGE\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_NEGOTIAEAUTHENTICATEDCHANNELKEYEXCHANGE\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_OFFERRESOURCES\_INVALIDPRIORITY,  
D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONHANDLE\_OUTOFMEMORY,  
D3D11\_MESSAGE\_ID\_ACQUIREHANDLEFORCAPTURE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_ACQUIREHANDLEFORCAPTURE\_INVALIDTYPE,  
D3D11\_MESSAGE\_ID\_ACQUIREHANDLEFORCAPTURE\_INVALIDBIND,  
D3D11\_MESSAGE\_ID\_ACQUIREHANDLEFORCAPTURE\_INVALIDARRAY,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMROTATION\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMROTATION\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMROTATION\_INVALID,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMROTATION\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMROTATION\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_INVALIDVIEW,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEVERTEXSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEVERTEXSHADER\_SHADEREXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEHULLSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEHULLSHADER\_SHADEREXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEDOMAINSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEDOMAINSHADER\_SHADEREXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADER\_SHADEREXTENSIONSNOTSUPPORTED,

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_DOUBLEEXTENSION

SNOTSUPPORTED,

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_SHADEREXTENSION  
SNOTSUPPORTED,

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEPIXELSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEPIXELSHADER\_SHADEREXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATECOMPUTESHADER\_DOUBLEEXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATECOMPUTESHADER\_SHADEREXTENSIONSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_SHADER\_LINKAGE\_MINPRECISION,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMALPHA\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPIXELASPECTRATIO\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEVERTEXSHADER\_UAVSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEHULLSHADER\_UAVSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEDOMAINSHADER\_UAVSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADER\_UAVSNOTSUPPORTED,

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_UAVSNOTSUPPORT  
ED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATEPIXELSHADER\_UAVSNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_CREATECOMPUTESHADER\_UAVSNOTSUPPORTED,

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS\_INVALIDOFF  
SET,

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS\_TOOMANYVIE  
WS,  
D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_NOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_SWAPDEVICECONTEXTSTATE\_NOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_UPDATESUBRESOURCE\_PREFERUPDATESUBRESOURCE1,  
D3D11\_MESSAGE\_ID\_GETDC\_INACCESSIBLE,  
D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_INVALIDRECT,  
D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_SAMPLE\_MASK\_IGNORED\_ON\_FL9,  
D3D11\_MESSAGE\_ID\_DEVICE\_OPEN\_SHARED\_RESOURCE1\_NOT\_SUPPORTED,  
D3D11\_MESSAGE\_ID\_DEVICE\_OPEN\_SHARED\_RESOURCE\_BY\_NAME\_NOT\_SUPPORTED,  
D3D11\_MESSAGE\_ID\_ENQUEUESETEVENT\_NOT\_SUPPORTED,  
D3D11\_MESSAGE\_ID\_OFFERRELEASE\_NOT\_SUPPORTED,  
D3D11\_MESSAGE\_ID\_OFFERRESOURCES\_INACCESSIBLE,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDMSAA,  
D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDMSAA,  
D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_INVALIDSOURCERECT,  
D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_EMPTYRECT,  
D3D11\_MESSAGE\_ID\_UPDATESUBRESOURCE\_EMPTYDESTBOX,  
D3D11\_MESSAGE\_ID\_COPYSUBRESOURCEREGION\_EMPTYSOURCEBOX,  
D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_OM\_RENDER\_TARGET\_DOES\_NOT\_SUPPORT\_LOGIC\_OPS,  
D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_DEPTHSTENCILVIEW\_NOT\_SET,  
D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RENDERTARGETVIEW\_NOT\_SET,  
D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RENDERTARGETVIEW\_NOT\_SET\_DUE\_TO\_FLIP\_PRESENT,  
D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_NOT\_SET\_DUE\_TO\_FLIP\_PRESENT,  
D3D11\_MESSAGE\_ID\_GETDATAFORNEWHARDWAREKEY\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CHECKCRYPTOSESSIONSTATUS\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONPRIVATEDATASIZE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETVIDEODECODERCAPS\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_GETVIDEODECODERCAPS\_ZEROWIDTHHEIGHT,  
D3D11\_MESSAGE\_ID\_CHECKVIDEODECODERDOWNSAMPLING\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_CHECKVIDEODECODERDOWNSAMPLING\_INVALIDCOLORSPACE,

D3D11\_MESSAGE\_ID\_CHECKVIDEODECODERDOWNSAMPLING\_ZEROWIDTHHEIGHT,  
D3D11\_MESSAGE\_ID\_VIDEODECODERENABLEDOWNSAMPLING\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEODECODERENABLEDOWNSAMPLING\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEODECODERUPDATEDOWNSAMPLING\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEODECODERUPDATEDOWNSAMPLING\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_CHECKVIDEOPROCESSORFORMATCONVERSION\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTCOLORSPACE1\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTCOLORSPACE1\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMCOLORSPACE1\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMCOLORSPACE1\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMMIRROR\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMMIRROR\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMMIRROR\_UNSUPPORTED,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMCOLORSPACE1\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMMIRROR\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_RECOMMENDVIDEODECODERDOWNSAMPLING\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_RECOMMENDVIDEODECODERDOWNSAMPLING\_INVALIDCOLORSPACE,  
D3D11\_MESSAGE\_ID\_RECOMMENDVIDEODECODERDOWNSAMPLING\_ZEROWIDTHHEIGHT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTSHADERUSAGE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTSHADERUSAGE\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETBEHAVIORHINTS\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETBEHAVIORHINTS\_INVALIDSTREAMCOUNT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETBEHAVIORHINTS\_TARGETRECT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETBEHAVIORHINTS\_INVALIDSOURCERECT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETBEHAVIORHINTS\_INVALIDDESTRECT,

D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONPRIVATEDATASIZE\_INVALID\_KEY\_EXCHANGE\_TYPE,  
D3D11\_MESSAGE\_ID\_DEVICE\_OPEN\_SHARED\_RESOURCE1\_ACCESS\_DENIED,  
D3D11\_MESSAGE\_ID\_D3D11\_1\_MESSAGES\_END,  
D3D11\_MESSAGE\_ID\_D3D11\_2\_MESSAGES\_START,  
D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDUSAGE,  
D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDUSAGE,  
D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDUSAGE,  
D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_LEVEL9\_STEPRATE\_NOT\_1,  
D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_LEVEL9\_INSTANCING\_NOT\_SUPPORTED,  
D3D11\_MESSAGE\_ID\_UPDATETILEMAPPINGS\_INVALID\_PARAMETER,  
D3D11\_MESSAGE\_ID\_COPYTILEMAPPINGS\_INVALID\_PARAMETER,  
D3D11\_MESSAGE\_ID\_COPYTILES\_INVALID\_PARAMETER,  
D3D11\_MESSAGE\_ID\_UPDATETILES\_INVALID\_PARAMETER,  
D3D11\_MESSAGE\_ID\_RESIZETILEPOOL\_INVALID\_PARAMETER,  
D3D11\_MESSAGE\_ID\_TILEDRESOURCEBARRIER\_INVALID\_PARAMETER,  
D3D11\_MESSAGE\_ID\_NULL\_TILE\_MAPPING\_ACCESS\_WARNING,  
D3D11\_MESSAGE\_ID\_NULL\_TILE\_MAPPING\_ACCESS\_ERROR,  
D3D11\_MESSAGE\_ID\_DIRTY\_TILE\_MAPPING\_ACCESS,  
D3D11\_MESSAGE\_ID\_DUPLICATE\_TILE\_MAPPINGS\_IN\_COVERED\_AREA,  
D3D11\_MESSAGE\_ID\_TILE\_MAPPINGS\_IN\_COVERED\_AREA\_DUPLICATED\_OUTSIDE,  
D3D11\_MESSAGE\_ID\_TILE\_MAPPINGS\_SHARED\_BETWEEN\_INCOMPATIBLE\_RESOURCES,  
D3D11\_MESSAGE\_ID\_TILE\_MAPPINGS\_SHARED\_BETWEEN\_INPUT\_AND\_OUTPUT,  
D3D11\_MESSAGE\_ID\_CHECKMULTISAMPLEQUALITYLEVELS\_INVALIDFLAGS,  
D3D11\_MESSAGE\_ID\_GETRESOURCETILING\_NONTILED\_RESOURCE,

D3D11\_MESSAGE\_ID\_RESIZETILEPOOL\_SHRINK\_WITH\_MAPPINGS\_STILL\_DEFINED\_PAST\_END,  
D3D11\_MESSAGE\_ID\_NEED\_TO\_CALL\_TILEDRESOURCEBARRIER,  
D3D11\_MESSAGE\_ID\_CREATEDevice\_INVALIDARGS,  
D3D11\_MESSAGE\_ID\_CREATEDevice\_WARNING,

D3D11\_MESSAGE\_ID\_CLEARUNORDEREDACCESSVIEWINT\_HAZARD,  
D3D11\_MESSAGE\_ID\_CLEARUNORDEREDACCESSVIEWFLOAT\_HAZARD,  
D3D11\_MESSAGE\_ID\_TILED\_RESOURCE\_TIER\_1\_BUFFER\_TEXTURE\_MISMATCH,  
D3D11\_MESSAGE\_ID\_CREATE\_CRYPTOSESSION,  
D3D11\_MESSAGE\_ID\_CREATE\_AUTHENTICATEDCHANNEL,  
D3D11\_MESSAGE\_ID\_LIVE\_CRYPTOSESSION,  
D3D11\_MESSAGE\_ID\_LIVE\_AUTHENTICATEDCHANNEL,  
D3D11\_MESSAGE\_ID\_DESTROY\_CRYPTOSESSION,  
D3D11\_MESSAGE\_ID\_DESTROY\_AUTHENTICATEDCHANNEL,  
D3D11\_MESSAGE\_ID\_D3D11\_2\_MESSAGES\_END,  
D3D11\_MESSAGE\_ID\_D3D11\_3\_MESSAGES\_START,  
D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_INVALID\_CONSERVATIVERASTERMODE,  
D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INVALID\_SYSTEMVALUE,  
D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_INVALIDCONTEXTTYPE,  
D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_DECODENOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_ENCODENOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDPLANEINDEX,  
D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDVIDEOPLANEINDEX,  
D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_AMBIGUOUSVIDEOPLANEINDEX,  
D3D11\_MESSAGE\_ID\_CREATERENDERTARGETVIEW\_INVALIDPLANEINDEX,  
D3D11\_MESSAGE\_ID\_CREATERENDERTARGETVIEW\_INVALIDVIDEOPLANEINDEX,  
D3D11\_MESSAGE\_ID\_CREATERENDERTARGETVIEW\_AMBIGUOUSVIDEOPLANEINDEX,  
D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDPLANEINDEX,  
D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDVIDEOPLANEINDEX,  
D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_AMBIGUOUSVIDEOPLANEINDEX,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDSCANDATAOFFSET,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_NOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_DIMENSIONSTOOLARGE,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDCOMPONENTS,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_DESTINATIONNOT2D,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_TILEDRESOURCESUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_GUARDRECTSUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_FORMATUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDSUBRESOURCE,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDMIPLEVEL,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_EMPTYDESTBOX,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_DESTBOXNOT2D,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_DESTBOXNOTSUB,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_DESTBOXESINTERSECT,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_XSAMPLEMISMATCH,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_YSAMPLEMISMATCH,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_XSAMPLEODD,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_YSAMPLEODD,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_OUTPUTDIMENSIONSTOOLARGE,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_NONPOW2SCALEUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_FRACTIONALDOWNSCALETOLARGE,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_CHROMASIZEMISMATCH,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_LUMACHROMASIZEMISMATCH,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDNUMDESTINATIONS,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_SUBBOXUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_1DESTUNSUPPORTEDFORMAT,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_3DESTUNSUPPORTEDFORMAT,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_SCALEUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDSOURCESIZE,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDCOPYFLAGS,

D3D11\_MESSAGE\_ID\_JPEGDECODE\_HAZARD,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_UNSUPPORTEDSRCBUFFERUSAGE,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_UNSUPPORTEDSRCBUFFERMISCFLAGS,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_UNSUPPORTEDDSTTEXTUREUSAGE,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_BACKBUFFERNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGDECODE\_UNSUPPRTEDCOPYFLAGS,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_NOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_INVALIDSCANDATAOFFSET,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_INVALIDCOMPONENTS,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_SOURCENOT2D,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_TILEDRESOURCESUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_GUARDRECTSUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_XSUBSAMPLEMISMATCH,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_YSUBSAMPLEMISMATCH,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_FORMATUNSUPPORTED,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_INVALIDSUBRESOURCE,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_INVALIDMIPLEVEL,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_DIMENSIONSTOOLARGE,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_HAZARD,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_UNSUPPORTEDDSTBUFFERUSAGE,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_UNSUPPORTEDDSTBUFFERMISCFLAGS,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_UNSUPPORTEDSRCTEXTUREUSAGE,  
D3D11\_MESSAGE\_ID\_JPEGENCODE\_BACKBUFFERNOTSUPPORTED,  
D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_UNSUPPORTEDCONTEXTTYPEFORQUERY,  
D3D11\_MESSAGE\_ID\_FLUSH1\_INVALIDCONTEXTTYPE,  
D3D11\_MESSAGE\_ID\_DEVICE\_SETHARDWAREPROTECTION\_INVALIDCONTEXT,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTHDRMETADATA\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTHDRMETADATA\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTHDRMETADATA\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTHDRMETADATA\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMHDRMETADATA\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMHDRMETADATA\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMHDRMETADATA\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMHDRMETADATA\_NULLPARAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMHDRMETADATA\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMHDRMETADATA\_INVALIDSIZE,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMFRAMEFORMAT\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMCOLORSPACE\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMOUTPUTRATE\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMSOURCERECT\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMDESTRECT\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMALPHA\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMPALETTE\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMPIXELASPECTRATIO\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMLUMAKEY\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMSTEREOFORMAT\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMAUTOPROCESSINGMODE\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMFILTER\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMROTATION\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMCOLORSPACE1\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMMIRROR\_INVALIDSTREAM,  
D3D11\_MESSAGE\_ID\_CREATE\_FENCE,  
D3D11\_MESSAGE\_ID\_LIVE\_FENCE,  
D3D11\_MESSAGE\_ID\_DESTROY\_FENCE,  
D3D11\_MESSAGE\_ID\_CREATE\_SYNCHRONIZEDCHANNEL,

```

D3D11_MESSAGE_ID_LIVE_SYNCHRONIZEDCHANNEL,
D3D11_MESSAGE_ID_DESTROY_SYNCHRONIZEDCHANNEL,
D3D11_MESSAGE_ID_CREATEFENCE_INVALIDFLAGS,
D3D11_MESSAGE_ID_D3D11_3_MESSAGES_END,
D3D11_MESSAGE_ID_D3D11_5_MESSAGES_START,

D3D11_MESSAGE_ID_NEGOTIATECRYPTOSESSIONKEYEXCHANGEMT_INVALIDKEYEXCHANGETYPE,
D3D11_MESSAGE_ID_NEGOTIATECRYPTOSESSIONKEYEXCHANGEMT_NOT_SUPPORTED,
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_COMPONENT_COUNT,
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_COMPONENT,
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_BUFFER_SIZE,
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_BUFFER_USAGE,
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_BUFFER_MISC_FLAGS,
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_BUFFER_OFFSET,
D3D11_MESSAGE_ID_CREATE_TRACKEDWORKLOAD,
D3D11_MESSAGE_ID_LIVE_TRACKEDWORKLOAD,
D3D11_MESSAGE_ID_DESTROY_TRACKEDWORKLOAD,
D3D11_MESSAGE_ID_CREATE_TRACKED_WORKLOAD_NULLPARAM,
D3D11_MESSAGE_ID_CREATE_TRACKED_WORKLOAD_INVALID_MAX_INSTANCES,
D3D11_MESSAGE_ID_CREATE_TRACKED_WORKLOAD_INVALID_DEADLINE_TYPE,
D3D11_MESSAGE_ID_CREATE_TRACKED_WORKLOAD_INVALID_ENGINE_TYPE,
D3D11_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOADS,
D3D11_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOAD_PAIRS,
D3D11_MESSAGE_ID_INCOMPLETE_TRACKED_WORKLOAD_PAIR,
D3D11_MESSAGE_ID_OUT_OF_ORDER_TRACKED_WORKLOAD_PAIR,
D3D11_MESSAGE_ID_CANNOT_ADD_TRACKED_WORKLOAD,
D3D11_MESSAGE_ID_TRACKED_WORKLOAD_NOT_SUPPORTED,
D3D11_MESSAGE_ID_TRACKED_WORKLOAD_ENGINE_TYPE_NOT_FOUND,
D3D11_MESSAGE_ID_NO_TRACKED_WORKLOAD_SLOT_AVAILABLE,
D3D11_MESSAGE_ID_END_TRACKED_WORKLOAD_INVALID_ARG,
D3D11_MESSAGE_ID_TRACKED_WORKLOAD_DISJOINT_FAILURE,
D3D11_MESSAGE_ID_D3D11_5_MESSAGES_END
} ;

```

## Constants

D3D11_MESSAGE_ID_UNKNOWN
Value: 0
D3D11_MESSAGE_ID_DEVICE_IASETVERTEXBUFFERS_HAZARD
D3D11_MESSAGE_ID_DEVICE_IASETINDEXBUFFER_HAZARD
D3D11_MESSAGE_ID_DEVICE_VSSETSHADERRESOURCES_HAZARD
D3D11_MESSAGE_ID_DEVICE_VSSETCONSTANTBUFFERS_HAZARD
D3D11_MESSAGE_ID_DEVICE_GSSETSHADERRESOURCES_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_GSSETCONSTANTBUFFERS\_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_PSSETSHADERRESOURCES\_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_PSSETCONSTANTBUFFERS\_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETS\_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_SOSETTARGETS\_HAZARD

D3D11\_MESSAGE\_ID\_STRING\_FROM\_APPLICATION

D3D11\_MESSAGE\_ID\_CORRUPTED\_THIS

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER1

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER2

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER3

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER4

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETERS5

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER6

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER7

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER8

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER9

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER10

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER11

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER12

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER13

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER14

D3D11\_MESSAGE\_ID\_CORRUPTED\_PARAMETER15

D3D11\_MESSAGE\_ID\_CORRUPTED\_MULTITHREADING

D3D11\_MESSAGE\_ID\_MESSAGE\_REPORTING\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_IASETINPUTLAYOUT\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_IASETVERTEXBUFFERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_IASETINDEXBUFFER\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_VSSETSHADER\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_VSSETSHADERRESOURCES\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_VSSETCONSTANTBUFFERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_VSSETSAMPLERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_GSSETSHADER\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_GSSETSHADERRESOURCES\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_GSSETCONSTANTBUFFERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_GSSETSAMPLERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_SOSETTARGETS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_PSSETSHADER\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_PSSETSHADERRESOURCES\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_PSSETCONSTANTBUFFERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_PSSETSAMPLERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_RSSETSTATE\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_OMSETBLENDSTATE\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_OMSETDEPTHSTENCILSTATE\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_OMSETRENDERTARGETS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_SETPREDICATION\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_GETPRIVATEDATA\_MOREDATA

D3D11\_MESSAGE\_ID\_SETPRIVATEDATA\_INVALIDFREEDATA

D3D11\_MESSAGE\_ID\_SETPRIVATEDATA\_INVALIDUNKNOWN

D3D11\_MESSAGE\_ID\_SETPRIVATEDATA\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_SETPRIVATEDATA\_CHANGINGPARAMS

D3D11\_MESSAGE\_ID\_SETPRIVATEDATA\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_UNRECOGNIZEDFORMAT

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDSAMPLES

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_UNRECOGNIZEDUSAGE

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_UNRECOGNIZEDBINDFLAGS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_UNRECOGNIZEDCPUACCESSFLAGS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_UNRECOGNIZEDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDCPUACCESSFLAGS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDBINDFLAGS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDINITIALDATA

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDMIPLEVELS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_NULLDESC

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDCONSTANTBUFFERBINDINGS

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_LARGEALLOCATION

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_UNRECOGNIZEDFORMAT

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_UNSUPPORTEDFORMAT

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDSAMPLES

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_UNRECOGNIZEDUSAGE

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_UNRECOGNIZEDBINDFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_UNRECOGNIZEDCPUACCESSFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_UNRECOGNIZEDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDCPUACCESSFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDBINDFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDINITIALDATA

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDMIPLEVELS

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_NULLDESC

D3D11\_MESSAGE\_ID\_CREATETEXTURE1D\_LARGEALLOCATION

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_UNRECOGNIZEDFORMAT

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_UNSUPPORTEDFORMAT

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDSAMPLES

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_UNRECOGNIZEDUSAGE

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_UNRECOGNIZEDBINDFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_UNRECOGNIZEDCPUACCESSFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_UNRECOGNIZEDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDCPUACCESSFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDBINDFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDINITIALDATA

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDMIPLEVELS

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_NULLDESC

D3D11\_MESSAGE\_ID\_CREATETEXTURE2D\_LARGEALLOCATION

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_UNRECOGNIZEDFORMAT

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_UNSUPPORTEDFORMAT

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_INVALIDSAMPLES

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_UNRECOGNIZEDUSAGE

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_UNRECOGNIZEDBINDFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_UNRECOGNIZEDCPUACCESSFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_UNRECOGNIZEDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_INVALIDCPUACCESSFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_INVALIDDBINDFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_INVALIDINITIALDATA

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_INVALIDMIPLEVELS

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_INVALIDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_NULLDESC

D3D11\_MESSAGE\_ID\_CREATETEXTURE3D\_LARGEALLOCATION

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_UNRECOGNIZEDFORMAT

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDDESC

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDRESOURCE

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_CREATE\_SHADERRESOURCEVIEW\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATE\_SHADERRESOURCEVIEW\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_UNRECOGNIZEDFORMAT

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_UNSUPPORTEDFORMAT

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_INVALIDDESC

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_INVALIDRESOURCE

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATE\_RENDER\_TARGETVIEW\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATE\_DEPTH\_STENCILVIEW\_UNRECOGNIZEDFORMAT

D3D11\_MESSAGE\_ID\_CREATE\_DEPTH\_STENCILVIEW\_INVALIDDESC

D3D11\_MESSAGE\_ID\_CREATE\_DEPTH\_STENCILVIEW\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_CREATE\_DEPTH\_STENCILVIEW\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_CREATE\_DEPTH\_STENCILVIEW\_INVALIDRESOURCE

D3D11\_MESSAGE\_ID\_CREATE\_DEPTH\_STENCILVIEW\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_CREATE\_DEPTH\_STENCILVIEW\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATE\_DEPTH\_STENCILVIEW\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATE\_INPUT\_LAYOUT\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_CREATE\_INPUT\_LAYOUT\_TOOMANYELEMENTS

D3D11\_MESSAGE\_ID\_CREATE\_INPUT\_LAYOUT\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_CREATE\_INPUT\_LAYOUT\_INCOMPATIBLEFORMAT

D3D11\_MESSAGE\_ID\_CREATE\_INPUT\_LAYOUT\_INVALIDSLOT

D3D11\_MESSAGE\_ID\_CREATE\_INPUT\_LAYOUT\_INVALIDINPUTSLOTCLASS

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_STEPRATESLOTCLASSMISMATCH

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_INVALIDSLOTCLASSCHANGE

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_INVALIDSTEPRATECHANGE

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_INVALIDALIGNMENT

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_DUPLICATESEMANTIC

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_UNPARSEABLEINPUTSIGNATURE

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_NULLSEMANTIC

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_MISSINGELEMENT

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_NULDESC

D3D11\_MESSAGE\_ID\_CREATEVERTEXSHADER\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_CREATEVERTEXSHADER\_INVALIDSHADERBYTECODE

D3D11\_MESSAGE\_ID\_CREATEVERTEXSHADER\_INVALIDSHADERTYPE

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADER\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADER\_INVALIDSHADERBYTECODE

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADER\_INVALIDSHADERTYPE

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDSHADERBYTECODE

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDSHADERTYPE

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDNUMENTRIES

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_OUTPUTSTREAMSTRIDEUNUSED

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_UNEXPECTEDDECL

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_EXPECTEDDECL

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_OUTPUTSLOT0EXPECTED

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDOUTPUTSLOT

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_ONLYONEELEMENTPERSLOT

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDCOMPONENTCOUNT

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDSTARTCOMPONENTANDCOMPONENTCOUNT

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDGAPDEFINITION

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_REPEATEDOUTPUT

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDOUTPUTSTREAMSTRIDE

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_MISSINGSEMANTIC

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_MASKMISMATCH

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_CANTHAVEONLYGAPS

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_DECLTOOCOMPLEX

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_MISSINGOUTPUTSIGNATURE

D3D11\_MESSAGE\_ID\_CREATEPIXELSHADER\_OUTOFGMEMORY

D3D11\_MESSAGE\_ID\_CREATEPIXELSHADER\_INVALIDSHADERBYTECODE

D3D11\_MESSAGE\_ID\_CREATEPIXELSHADER\_INVALIDSHADERTYPE

D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_INVALIDFILLMODE

D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_INVALIDCULLMODE

D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_INVALIDDEPTHBIASCLAMP

D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_INVALIDSLOPESCALEDDEPTHBIAS

D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_NULLDESC

D3D11\_MESSAGE\_ID\_CREATEDEPTHSTENCILSTATE\_INVALIDDEPTHWRITEMASK

D3D11\_MESSAGE\_ID\_CREATEDEPTHSTENCILSTATE\_INVALIDDEPTHFUNC

D3D11\_MESSAGE\_ID\_CREATEDEPTHSTENCILSTATE\_INVALIDFRONTFACE\_STENCILFAILOP

D3D11\_MESSAGE\_ID\_CREATEDEPTHSTENCILSTATE\_INVALIDFRONTFACE\_STENCILZFAILOP

D3D11\_MESSAGE\_ID\_CREATEDEPTHSTENCILSTATE\_INVALIDFRONTFACE\_STENCILPASSOP

D3D11\_MESSAGE\_ID\_CREATEDEPTHSTENCILSTATE\_INVALIDFRONTFACE\_STENCILFUNC

D3D11\_MESSAGE\_ID\_CREATEDEPTH\_STENCILSTATE\_INVALIDBACKFACE\_STENCILFAIL\_OPS

D3D11\_MESSAGE\_ID\_CREATEDEPTH\_STENCILSTATE\_INVALIDBACKFACE\_STENCILZFAIL\_OPS

D3D11\_MESSAGE\_ID\_CREATEDEPTH\_STENCILSTATE\_INVALIDBACKFACE\_STENCILPASS\_OPS

D3D11\_MESSAGE\_ID\_CREATEDEPTH\_STENCILSTATE\_INVALIDBACKFACE\_STENCILFUNC

D3D11\_MESSAGE\_ID\_CREATEDEPTH\_STENCILSTATE\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_CREATEDEPTH\_STENCILSTATE\_NULLDESC

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDSRCBLEND

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDDESTBLEND

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDBLENDOP

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDSRCBLENDALPHA

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDDESTBLENDALPHA

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDBLENDOPALPHA

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDRENDERTARGETWRITEMASK

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_NULLDESC

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDFILTER

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDADDRESS\_U

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDADDRESS\_V

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDADDRESS\_W

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDMIPLODBIAS

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDMAXANISOTROPY

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDCOMPARISONFUNC

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDMINLOD

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_INVALIDMAXLOD

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_NULLDESC

D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_INVALIDIDQUERY

D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_INVALIDMISCFLAGS

D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_UNEXPECTEDMISCFLAG

D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_NULLDESC

D3D11\_MESSAGE\_ID\_DEVICE\_IASETPRIMITIVETOPOLOGY\_TOPOLOGY\_UNRECOGNIZED

D3D11\_MESSAGE\_ID\_DEVICE\_IASETPRIMITIVETOPOLOGY\_TOPOLOGY\_UNDEFINED

D3D11\_MESSAGE\_ID\_IASETVERTEXBUFFERS\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_IASETVERTEXBUFFERS\_OFFSET\_TOO\_LARGE

D3D11\_MESSAGE\_ID\_DEVICE\_IASETVERTEXBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_IASETINDEXBUFFER\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_IASETINDEXBUFFER\_FORMAT\_INVALID

D3D11\_MESSAGE\_ID\_DEVICE\_IASETINDEXBUFFER\_OFFSET\_TOO\_LARGE

D3D11\_MESSAGE\_ID\_DEVICE\_IASETINDEXBUFFER\_OFFSET\_UNALIGNED

D3D11\_MESSAGE\_ID\_DEVICE\_VSSETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_VSSETCONSTANTBUFFERS\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_VSSETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_VSSETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_GSSETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_GSSETCONSTANTBUFFERS\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_GSSETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_GSSETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_SOSETTARGETS\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_SOSETTARGETS\_OFFSET\_UNALIGNED

D3D11\_MESSAGE\_ID\_DEVICE\_PSSETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_PSSETCONSTANTBUFFERS\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_PSSETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_PSSETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_RSSETVIEWPORTS\_INVALIDVIEWPORT

D3D11\_MESSAGE\_ID\_DEVICE\_RSSETSCISSORRECTS\_INVALIDSCISSOR

D3D11\_MESSAGE\_ID\_CLEARRENDERTARGETVIEW\_DENORMFLUSH

D3D11\_MESSAGE\_ID\_CLEARDEPTHSTENCILVIEW\_DENORMFLUSH

D3D11\_MESSAGE\_ID\_CLEARDEPTHSTENCILVIEW\_INVALID

D3D11\_MESSAGE\_ID\_DEVICE\_IAGETVERTEXBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_VSGETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_VSGETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_VSGETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_GSGETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_GSGETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_GSGETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_SOGETTARGETS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_PSGETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_PSGETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_PSGETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_RSGETVIEWPORTS\_VIEWPORTS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_RSGETSCISSORRECTS\_RECTS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_GENERATEMIPS\_RESOURCE\_INVALID

D3D11\_MESSAGE\_ID\_COPYSUBRESOURCEREGION\_INVALIDDESTINATIONSUBRESOURCE

D3D11\_MESSAGE\_ID\_COPYSUBRESOURCEREGION\_INVALIDSOURCESUBRESOURCE

D3D11\_MESSAGE\_ID\_COPYSUBRESOURCEREGION\_INVALIDSOURCEBOX

D3D11\_MESSAGE\_ID\_COPYSUBRESOURCEREGION\_INVALIDSOURCE

D3D11\_MESSAGE\_ID\_COPYSUBRESOURCEREGION\_INVALIDDESTINATIONSTATE

D3D11\_MESSAGE\_ID\_COPYSUBRESOURCEREGION\_INVALIDSOURCESTATE

D3D11\_MESSAGE\_ID\_COPYRESOURCE\_INVALIDSOURCE

D3D11\_MESSAGE\_ID\_COPYRESOURCE\_INVALIDDESTINATIONSTATE

D3D11\_MESSAGE\_ID\_COPYRESOURCE\_INVALIDSOURCESTATE

D3D11\_MESSAGE\_ID\_UPDATESUBRESOURCE\_INVALIDDESTINATIONSUBRESOURCE

D3D11\_MESSAGE\_ID\_UPDATESUBRESOURCE\_INVALIDDESTINATIONBOX

D3D11\_MESSAGE\_ID\_UPDATESUBRESOURCE\_INVALIDDESTINATIONSTATE

D3D11\_MESSAGE\_ID\_DEVICE\_RESOLVESUBRESOURCE\_DESTINATION\_INVALID

D3D11\_MESSAGE\_ID\_DEVICE\_RESOLVESUBRESOURCE\_DESTINATION\_SUBRESOURCE\_INVALID

D3D11\_MESSAGE\_ID\_DEVICE\_RESOLVESUBRESOURCE\_SOURCE\_INVALID

D3D11\_MESSAGE\_ID\_DEVICE\_RESOLVESUBRESOURCE\_SOURCE\_SUBRESOURCE\_INVALID

D3D11\_MESSAGE\_ID\_DEVICE\_RESOLVESUBRESOURCE\_FORMAT\_INVALID

D3D11\_MESSAGE\_ID\_BUFFER\_MAP\_INVALIDMAPTYPE

D3D11\_MESSAGE\_ID\_BUFFER\_MAP\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_BUFFER\_MAP\_ALREADYMAPPED

D3D11\_MESSAGE\_ID\_BUFFER\_MAP\_DEVICEREMOVED\_RETURN

D3D11\_MESSAGE\_ID\_BUFFER\_UNMAP\_NOTMAPPED

D3D11\_MESSAGE\_ID\_TEXTURE1D\_MAP\_INVALIDMAPTYPE

D3D11\_MESSAGE\_ID\_TEXTURE1D\_MAP\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_TEXTURE1D\_MAP\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_TEXTURE1D\_MAP\_ALREADYMAPPED

D3D11\_MESSAGE\_ID\_TEXTURE1D\_MAP\_DEVICEREMOVED\_RETURN

D3D11\_MESSAGE\_ID\_TEXTURE1D\_UNMAP\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_TEXTURE1D\_UNMAP\_NOTMAPPED

D3D11\_MESSAGE\_ID\_TEXTURE2D\_MAP\_INVALIDMAPTYPE

D3D11\_MESSAGE\_ID\_TEXTURE2D\_MAP\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_TEXTURE2D\_MAP\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_TEXTURE2D\_MAP\_ALREADYMAPPED

D3D11\_MESSAGE\_ID\_TEXTURE2D\_MAP\_DEVICEREMOVED\_RETURN

D3D11\_MESSAGE\_ID\_TEXTURE2D\_UNMAP\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_TEXTURE2D\_UNMAP\_NOTMAPPED

D3D11\_MESSAGE\_ID\_TEXTURE3D\_MAP\_INVALIDMAPTYPE

D3D11\_MESSAGE\_ID\_TEXTURE3D\_MAP\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_TEXTURE3D\_MAP\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_TEXTURE3D\_MAP\_ALREADYMAPPED

D3D11\_MESSAGE\_ID\_TEXTURE3D\_MAP\_DEVICEREMOVED\_RETURN

D3D11\_MESSAGE\_ID\_TEXTURE3D\_UNMAP\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_TEXTURE3D\_UNMAP\_NOTMAPPED

D3D11\_MESSAGE\_ID\_CHECKFORMATSUPPORT\_FORMAT\_DEPRECATED

D3D11\_MESSAGE\_ID\_CHECKMULTISAMPLEQUALITYLEVELS\_FORMAT\_DEPRECATED

D3D11\_MESSAGE\_ID\_SETEXCEPTIONMODE\_UNRECOGNIZEDFLAGS

D3D11\_MESSAGE\_ID\_SETEXCEPTIONMODE\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_SETEXCEPTIONMODE\_DEVICEREMOVED\_RETURN

D3D11\_MESSAGE\_ID\_REF\_SIMULATING\_INFINITELY\_FAST\_HARDWARE

D3D11\_MESSAGE\_ID\_REF\_THREADING\_MODE

D3D11\_MESSAGE\_ID\_REF\_UMDRIVER\_EXCEPTION

D3D11\_MESSAGE\_ID\_REF\_KMDRIVER\_EXCEPTION

D3D11\_MESSAGE\_ID\_REF\_HARDWARE\_EXCEPTION

D3D11\_MESSAGE\_ID\_REF\_ACCESSING\_INDEXABLE\_TEMP\_OUT\_OF\_RANGE

D3D11\_MESSAGE\_ID\_REF\_PROBLEM\_PARSING\_SHADER

D3D11\_MESSAGE\_ID\_REF\_OUT\_OF\_MEMORY

D3D11\_MESSAGE\_ID\_REF\_INFO

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VERTEXPOS\_OVERFLOW

D3D11\_MESSAGE\_ID\_DEVICE\_DRAWINDEXED\_INDEXPOS\_OVERFLOW

D3D11\_MESSAGE\_ID\_DEVICE\_DRAWINSTANCED\_VERTEXPOS\_OVERFLOW

D3D11\_MESSAGE\_ID\_DEVICE\_DRAWINSTANCED\_INSTANCEPOS\_OVERFLOW

D3D11\_MESSAGE\_ID\_DEVICE\_DRAWINDEXEDINSTANCED\_INSTANCEPOS\_OVERFLOW

D3D11\_MESSAGE\_ID\_DEVICE\_DRAWINDEXEDINSTANCED\_INDEXPOS\_OVERFLOW

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VERTEX\_SHADER\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_SHADER\_LINKAGE\_SEMANTICNAME\_NOT\_FOUND

D3D11\_MESSAGE\_ID\_DEVICE\_SHADER\_LINKAGE\_REGISTERINDEX

D3D11\_MESSAGE\_ID\_DEVICE\_SHADER\_LINKAGE\_COMPONENTTYPE

D3D11\_MESSAGE\_ID\_DEVICE\_SHADER\_LINKAGE\_REGISTERMASK

D3D11\_MESSAGE\_ID\_DEVICE\_SHADER\_LINKAGE\_SYSTEMVALUE

D3D11\_MESSAGE\_ID\_DEVICE\_SHADER\_LINKAGE\_NEVERWRITTEN\_ALWAYSREADS

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VERTEX\_BUFFER\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INPUTLAYOUT\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_CONSTANT\_BUFFER\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_CONSTANT\_BUFFER\_TOO\_SMALL

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_SAMPLER\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_SHADERRESOURCEVIEW\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VIEW\_DIMENSION\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VERTEX\_BUFFER\_STRIDE\_TOO\_SMALL

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VERTEX\_BUFFER\_TOO\_SMALL

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INDEX\_BUFFER\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INDEX\_BUFFER\_FORMAT\_INVALID

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INDEX\_BUFFER\_TOO\_SMALL

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_GS\_INPUT\_PRIMITIVE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RESOURCE\_RETURN\_TYPE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_POSITION\_NOT\_PRESENT

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_OUTPUT\_STREAM\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_BOUND\_RESOURCE\_MAPPED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INVALID\_PRIMITIVE\_TOPOLOGY

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VERTEX\_OFFSET\_UNALIGNED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VERTEX\_STRIDE\_UNALIGNED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INDEX\_OFFSET\_UNALIGNED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_OUTPUT\_STREAM\_OFFSET\_UNALIGNED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RESOURCE\_FORMAT\_LD\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RESOURCE\_FORMAT\_SAMPLE\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RESOURCE\_FORMAT\_SAMPLE\_C\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RESOURCE\_MULTISAMPLE\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_SO\_TARGETS\_BOUND\_WITHOUT\_SOURCE

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_SO\_STRIDE\_LARGER\_THAN\_BUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_OM\_RENDER\_TARGET\_DOES\_NOT\_SUPPORT\_BLENDING

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_OM\_DUAL\_SOURCE\_BLENDING\_CAN\_ONLY\_HAVE\_RENDER\_TARGET\_0

D3D11\_MESSAGE\_ID\_DEVICE\_REMOVAL\_PROCESS\_AT\_FAULT

D3D11\_MESSAGE\_ID\_DEVICE\_REMOVAL\_PROCESS\_POSSIBLY\_AT\_FAULT

D3D11\_MESSAGE\_ID\_DEVICE\_REMOVAL\_PROCESS\_NOT\_AT\_FAULT

D3D11\_MESSAGE\_ID\_DEVICE\_OPEN\_SHARED\_RESOURCE\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_DEVICE\_OPEN\_SHARED\_RESOURCE\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_DEVICE\_OPEN\_SHARED\_RESOURCE\_BADINTERFACE\_RETURN

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_VIEWPORT\_NOT\_SET

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_TRAILING\_DIGIT\_IN\_SEMANTIC

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_TRAILING\_DIGIT\_IN\_SEMANTIC

D3D11\_MESSAGE\_ID\_DEVICE\_RSSETVIEWPORTS\_DENORMFLUSH

D3D11\_MESSAGE\_ID\_OMSETRENDERTARGETS\_INVALIDVIEW

D3D11\_MESSAGE\_ID\_DEVICE\_SETTEXTFILTERSIZE\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_SAMPLER\_MISMATCH

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_TYPE\_MISMATCH

D3D11\_MESSAGE\_ID\_BLENDSTATE\_GETDESC\_LEGACY

D3D11\_MESSAGE\_ID\_SHADERRESOURCEVIEW\_GETDESC\_LEGACY

D3D11\_MESSAGE\_ID\_CREATEQUERY\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEPREDICATE\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATECOUNTER\_OUTOFRANGE\_COUNTER

D3D11\_MESSAGE\_ID\_CREATECOUNTER\_SIMULTANEOUS\_ACTIVE\_COUNTERS\_EXHAUSTED

D3D11\_MESSAGE\_ID\_CREATECOUNTER\_UNSUPPORTED\_WELLKNOWN\_COUNTER

D3D11\_MESSAGE\_ID\_CREATECOUNTER\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATECOUNTER\_NONEXCLUSIVE\_RETURN

D3D11\_MESSAGE\_ID\_CREATECOUNTER\_NULLDESC

D3D11\_MESSAGE\_ID\_CHECKCOUNTER\_OUTOFRANGE\_COUNTER

D3D11\_MESSAGE\_ID\_CHECKCOUNTER\_UNSUPPORTED\_WELLKNOWN\_COUNTER

D3D11\_MESSAGE\_ID\_SETPREDICATION\_INVALID\_PREDICATE\_STATE

D3D11\_MESSAGE\_ID\_QUERY\_BEGIN\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_PREDICATE\_BEGIN\_DURING\_PREDICATION

D3D11\_MESSAGE\_ID\_QUERY\_BEGIN\_DUPLICATE

D3D11\_MESSAGE\_ID\_QUERY\_BEGIN\_ABANDONING\_PREVIOUS\_RESULTS

D3D11\_MESSAGE\_ID\_PREDICATE\_END\_DURING\_PREDICATION

D3D11\_MESSAGE\_ID\_QUERY\_END\_ABANDONING\_PREVIOUS\_RESULTS

D3D11\_MESSAGE\_ID\_QUERY\_END\_WITHOUT\_BEGIN

D3D11\_MESSAGE\_ID\_QUERY\_GETDATA\_INVALID\_DATASIZE

D3D11\_MESSAGE\_ID\_QUERY\_GETDATA\_INVALID\_FLAGS

D3D11\_MESSAGE\_ID\_QUERY\_GETDATA\_INVALID\_CALL

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_PS\_OUTPUT\_TYPE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RESOURCE\_FORMAT\_GATHER\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INVALID\_USE\_OF\_CENTER\_MULTISAMPLE\_PATTERN

D3D11\_MESSAGE\_ID\_DEVICE\_IASETVERTEXBUFFERS\_STRIDE\_TOO\_LARGE

D3D11\_MESSAGE\_ID\_DEVICE\_IASETVERTEXBUFFERS\_INVALIDRANGE

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_EMPTY\_LAYOUT

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RESOURCE\_SAMPLE\_COUNT\_MISMATCH

D3D11\_MESSAGE\_ID\_LIVE\_OBJECT\_SUMMARY

D3D11\_MESSAGE\_ID\_LIVE\_BUFFER

D3D11\_MESSAGE\_ID\_LIVE\_TEXTURE1D

D3D11\_MESSAGE\_ID\_LIVE\_TEXTURE2D

D3D11\_MESSAGE\_ID\_LIVE\_TEXTURE3D

D3D11\_MESSAGE\_ID\_LIVE\_SHADERRESOURCEVIEW

D3D11\_MESSAGE\_ID\_LIVE\_RENDERTARGETVIEW

D3D11\_MESSAGE\_ID\_LIVE\_DEPTHSTENCILVIEW

D3D11\_MESSAGE\_ID\_LIVE\_VERTEXSHADER

D3D11\_MESSAGE\_ID\_LIVE\_GEOMETRYSHADER

D3D11\_MESSAGE\_ID\_LIVE\_PIXELSHADER

D3D11\_MESSAGE\_ID\_LIVE\_INPUTLAYOUT

D3D11\_MESSAGE\_ID\_LIVE\_SAMPLER

D3D11\_MESSAGE\_ID\_LIVE\_BLENDSTATE

D3D11\_MESSAGE\_ID\_LIVE\_DEPTHSTENCILSTATE

D3D11\_MESSAGE\_ID\_LIVE\_RASTERIZERSTATE

D3D11\_MESSAGE\_ID\_LIVE\_QUERY

D3D11\_MESSAGE\_ID\_LIVE\_PREDICATE

D3D11\_MESSAGE\_ID\_LIVE\_COUNTER

D3D11\_MESSAGE\_ID\_LIVE\_DEVICE

D3D11\_MESSAGE\_ID\_LIVE\_SWAPCHAIN

D3D11\_MESSAGE\_ID\_D3D10\_MESSAGES\_END

D3D11\_MESSAGE\_ID\_D3D10L9\_MESSAGES\_START

Value: 0x100000

D3D11\_MESSAGE\_ID\_CREATEDEPTHSTENCILSTATE\_STENCIL\_NO\_TWO\_SIDED

D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_DepthBiasClamp\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_NO\_COMPARISON\_SUPPORT

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_EXCESSIVE\_ANISOTROPY

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_BORDER\_OUT\_OF\_RANGE

D3D11\_MESSAGE\_ID\_VSSETSAMPLERS\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_VSSETSAMPLERS\_TOO\_MANY\_SAMPLERS

D3D11\_MESSAGE\_ID\_PSSETSAMPLERS\_TOO\_MANY\_SAMPLERS

D3D11\_MESSAGE\_ID\_CREATEROFILE\_NO\_ARRAYS

D3D11\_MESSAGE\_ID\_CREATEROFILE\_NO\_VB\_AND\_IB\_BIND

D3D11\_MESSAGE\_ID\_CREATEROFILE\_NO\_TEXTURE\_1D

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_DIMENSION\_OUT\_OF\_RANGE

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_NOT\_BINDABLE\_AS\_SHADER\_RESOURCE

D3D11\_MESSAGE\_ID\_OMSETRENDERTARGETS\_TOO\_MANY\_RENDER\_TARGETS

D3D11\_MESSAGE\_ID\_OMSETRENDERTARGETS\_NO\_DIFFERING\_BIT\_DEPTHS

D3D11\_MESSAGE\_ID\_IASETVERTEXBUFFERS\_BAD\_BUFFER\_INDEX

D3D11\_MESSAGE\_ID\_DEVICE\_RSSETVIEWPORTS\_TOO\_MANY\_VIEWPORTS

D3D11\_MESSAGE\_ID\_DEVICE\_IASETPRIMITIVETOPOLOGY\_ADJACENCY\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_RSSETSCISSORRECTS\_TOO\_MANY\_SCISSORS

D3D11\_MESSAGE\_ID\_COPYRESOURCE\_ONLY\_TEXTURE\_2D\_WITHIN\_GPU\_MEMORY

D3D11\_MESSAGE\_ID\_COPYRESOURCE\_NO\_TEXTURE\_3D\_READBACK

D3D11\_MESSAGE\_ID\_COPYRESOURCE\_NO\_TEXTURE\_ONLY\_READBACK

D3D11\_MESSAGE\_ID\_CREATEINPUTLAYOUT\_UNSUPPORTED\_FORMAT

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_NO\_ALPHA\_TO\_COVERAGE

D3D11\_MESSAGE\_ID\_CREATERASTERIZERSTATE\_DepthClipEnable\_MUST\_BE\_TRUE

D3D11\_MESSAGE\_ID\_DRAWINDEXED\_STARTINDEXLOCATION\_MUST\_BE\_POSITIVE

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_MUST\_USE\_LOWEST\_LOD

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_MINLOD\_MUST\_NOT\_BE\_FRACTIONAL

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_MAXLOD\_MUST\_BE\_FLT\_MAX

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_FIRSTARRAYSLICE\_MUST\_BE\_ZERO

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_CUBES\_MUST\_HAVE\_6\_SIDES

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_NOT\_BINDABLE\_AS\_RENDER\_TARGET

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_NO\_DWORD\_INDEX\_BUFFER

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_MSAA\_PRECLUDES\_SHADER\_RESOURCE

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_PRESENTATION\_PRECLUDES\_SHADER\_RESOURCE

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_NO\_INDEPENDENT\_BLEND\_ENABLE

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_NO\_INDEPENDENT\_WRITE\_MASKS

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_NO\_STREAM\_OUT

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_ONLY\_VB\_IB\_FOR\_BUFFERS

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_NO\_AUTOGEN\_FOR\_VOLUMES

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_DXGI\_FORMAT\_R8G8B8A8\_CANNOT\_BE\_SHARED

D3D11\_MESSAGE\_ID\_VSSHADERRESOURCES\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_GEOMETRY\_SHADER\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_STREAM\_OUT\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_TEXT\_FILTER\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_NO\_SEPARATE\_ALPHA\_BLEND

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_NO\_MRT\_BLEND

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_OPERATION\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_NO\_MIRRORONCE

D3D11\_MESSAGE\_ID\_DRAWINSTANCED\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_DRAWINDEXEDINSTANCED\_NOT\_SUPPORTED\_BELOW\_9\_3

D3D11\_MESSAGE\_ID\_DRAWINDEXED\_POINTLIST\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_SETBLENDSTATE\_SAMPLE\_MASK\_CANNOT\_BE\_ZERO

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_DIMENSION\_EXCEEDS\_FEATURE\_LEVEL\_DEFINITION

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_ONLY\_SINGLE\_MIP\_LEVEL\_DEPTH\_STENCIL\_SUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_RSSETSCISSORRECTS\_NEGATIVESCISSOR

D3D11\_MESSAGE\_ID\_SLOT\_ZERO\_MUST\_BE\_D3D10\_INPUT\_PER\_VERTEX\_DATA

D3D11\_MESSAGE\_ID\_CREATERESOURCE\_NON\_POW\_2\_MIPMAP

D3D11\_MESSAGE\_ID\_CREATESAMPLERSTATE\_BORDER\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_OMSETRENDERTARGETS\_NO\_SRGB\_MRT

D3D11\_MESSAGE\_ID\_COPYRESOURCE\_NO\_3D\_MISMATCHED\_UPDATES

D3D11\_MESSAGE\_ID\_D3D10L9\_MESSAGES\_END

D3D11\_MESSAGE\_ID\_D3D11\_MESSAGES\_START

Value: 0x200000

D3D11\_MESSAGE\_ID\_CREATEDEPTHSTENCILVIEW\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_CREATEVERTEXSHADER\_INVALIDCLASSLINKAGE

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADER\_INVALIDCLASSLINKAGE

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDNUMSTREAMS

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDSTREAMTORASTERIZER

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_UNEXPECTEDSTREAMS

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDCLASSLINKAGE

D3D11\_MESSAGE\_ID\_CREATEPIXELSHADER\_INVALIDCLASSLINKAGE

D3D11\_MESSAGE\_ID\_CREATEDFERREDCONTEXT\_INVALID\_COMMANDLISTFLAGS

D3D11\_MESSAGE\_ID\_CREATEDFERREDCONTEXT\_SINGLETHREADED

D3D11\_MESSAGE\_ID\_CREATEDFERREDCONTEXT\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATEDFERREDCONTEXT\_INVALID\_CALL\_RETURN

D3D11\_MESSAGE\_ID\_CREATEDFERREDCONTEXT\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_FINISHDISPLAYLIST\_ONIMMEDIATECONTEXT

D3D11\_MESSAGE\_ID\_FINISHDISPLAYLIST\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_FINISHDISPLAYLIST\_INVALID\_CALL\_RETURN

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_UNEXPECTEDENTRIES

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_UNEXPECTEDSTRIDES

D3D11\_MESSAGE\_ID\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_INVALIDNUMSTRIDES

D3D11\_MESSAGE\_ID\_DEVICE\_HSSETSHADERRESOURCES\_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_HSSETCONSTANTBUFFERS\_HAZARD

D3D11\_MESSAGE\_ID\_HSSETSHADERRESOURCES\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_HSSETCONSTANTBUFFERS\_UNBINDELETEOBJECT

D3D11\_MESSAGE\_ID\_CREATEHULLSHADER\_INVALIDCALL

D3D11\_MESSAGE\_ID\_CREATEHULLSHADER\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_CREATEHULLSHADER\_INVALIDSHADERBYTECODE

D3D11\_MESSAGE\_ID\_CREATEHULLSHADER\_INVALIDSHADERTYPE

D3D11\_MESSAGE\_ID\_CREATEHULLSHADER\_INVALIDCLASSLINKAGE

D3D11\_MESSAGE\_ID\_DEVICE\_HSSETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_HSSETCONSTANTBUFFERS\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_HSSETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_HSSETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_HSGETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_HSGETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_HSGETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_DSSETSHADERRESOURCES\_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_DSSETCONSTANTBUFFERS\_HAZARD

D3D11\_MESSAGE\_ID\_DSSETSHADERRESOURCES\_UNBINDELETEOBJECT

D3D11\_MESSAGE\_ID\_DSSETCONSTANTBUFFERS\_UNBINDELETEOBJECT

D3D11\_MESSAGE\_ID\_CREATEDOMAINSHADER\_INVALIDCALL

D3D11\_MESSAGE\_ID\_CREATEDOMAINSHADER\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_CREATEDOMAINSHADER\_INVALIDSHADERBYTECODE

D3D11\_MESSAGE\_ID\_CREATEDOMAINSHADER\_INVALIDSHADERTYPE

D3D11\_MESSAGE\_ID\_CREATEDOMAINSHADER\_INVALIDCLASSLINKAGE

D3D11\_MESSAGE\_ID\_DEVICE\_DSSETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_DSSETCONSTANTBUFFERS\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_DSSETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_DSSETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_DSGETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_DSGETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_DSGETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_HS\_XOR\_DS\_MISMATCH

D3D11\_MESSAGE\_ID\_DEFERRED\_CONTEXT\_REMOVAL\_PROCESS\_AT\_FAULT

D3D11\_MESSAGE\_ID\_DEVICE\_DRAWINDIRECT\_INVALID\_ARG\_BUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_DRAWINDIRECT\_OFFSET\_UNALIGNED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAWINDIRECT\_OFFSET\_OVERFLOW

D3D11\_MESSAGE\_ID\_RESOURCE\_MAP\_INVALIDMAPTYPE

D3D11\_MESSAGE\_ID\_RESOURCE\_MAP\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_RESOURCE\_MAP\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_RESOURCE\_MAP\_ALREADYMAPPED

D3D11\_MESSAGE\_ID\_RESOURCE\_MAP\_DEVICEREMOVED\_RETURN

D3D11\_MESSAGE\_ID\_RESOURCE\_MAP\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_RESOURCE\_MAP\_WITHOUT\_INITIAL\_DISCARD

D3D11\_MESSAGE\_ID\_RESOURCE\_UNMAP\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_RESOURCE\_UNMAP\_NOTMAPPED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RASTERIZING\_CONTROL\_POINTS

D3D11\_MESSAGE\_ID\_DEVICE\_IASETPRIMITIVETOPOLOGY\_TOPOLOGY\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_HS\_DS\_SIGNATURE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_HULL\_SHADER\_INPUT\_TOPOLOGY\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_HS\_DS\_CONTROL\_POINT\_COUNT\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_HS\_DS\_TESSELLATOR\_DOMAIN\_MISMATCH

D3D11\_MESSAGE\_ID\_CREATE\_CONTEXT

D3D11\_MESSAGE\_ID\_LIVE\_CONTEXT

D3D11\_MESSAGE\_ID\_DESTROY\_CONTEXT

D3D11\_MESSAGE\_ID\_CREATE\_BUFFER

D3D11\_MESSAGE\_ID\_LIVE\_BUFFER\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_BUFFER

D3D11\_MESSAGE\_ID\_CREATE\_TEXTURE1D

D3D11\_MESSAGE\_ID\_LIVE\_TEXTURE1D\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_TEXTURE1D

D3D11\_MESSAGE\_ID\_CREATE\_TEXTURE2D

D3D11\_MESSAGE\_ID\_LIVE\_TEXTURE2D\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_TEXTURE2D

D3D11\_MESSAGE\_ID\_CREATE\_TEXTURE3D

D3D11\_MESSAGE\_ID\_LIVE\_TEXTURE3D\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_TEXTURE3D

D3D11\_MESSAGE\_ID\_CREATE\_SHADERRESOURCEVIEW

D3D11\_MESSAGE\_ID\_LIVE\_SHADERRESOURCEVIEW\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_SHADERRESOURCEVIEW

D3D11\_MESSAGE\_ID\_CREATE\_RENDERTARGETVIEW

D3D11\_MESSAGE\_ID\_LIVE\_RENDERTARGETVIEW\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_RENDERTARGETVIEW

D3D11\_MESSAGE\_ID\_CREATE\_DEPTHSTENCILVIEW

D3D11\_MESSAGE\_ID\_LIVE\_DEPTHSTENCILVIEW\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_DEPTHSTENCILVIEW

D3D11\_MESSAGE\_ID\_CREATE\_VERTEXSHADER

D3D11\_MESSAGE\_ID\_LIVE\_VERTEXSHADER\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_VERTEXSHADER

D3D11\_MESSAGE\_ID\_CREATE\_HULLSHADER

D3D11\_MESSAGE\_ID\_LIVE\_HULLSHADER

D3D11\_MESSAGE\_ID\_DESTROY\_HULLSHADER

D3D11\_MESSAGE\_ID\_CREATE\_DOMAINSHADER

D3D11\_MESSAGE\_ID\_LIVE\_DOMAINSHADER

D3D11\_MESSAGE\_ID\_DESTROY\_DOMAINSHADER

D3D11\_MESSAGE\_ID\_CREATE\_GEOMETRYSHADER

D3D11\_MESSAGE\_ID\_LIVE\_GEOMETRYSHADER\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_GEOMETRYSHADER

D3D11\_MESSAGE\_ID\_CREATE\_PIXELSHADER

D3D11\_MESSAGE\_ID\_LIVE\_PIXELSHADER\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_PIXELSHADER

D3D11\_MESSAGE\_ID\_CREATE\_INPUTLAYOUT

D3D11\_MESSAGE\_ID\_LIVE\_INPUTLAYOUT\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_INPUTLAYOUT

D3D11\_MESSAGE\_ID\_CREATE\_SAMPLER

D3D11\_MESSAGE\_ID\_LIVE\_SAMPLER\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_SAMPLER

D3D11\_MESSAGE\_ID\_CREATE\_BLENDSTATE

D3D11\_MESSAGE\_ID\_LIVE\_BLENDSTATE\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_BLENDSTATE

D3D11\_MESSAGE\_ID\_CREATE\_DEPTHSTENCILSTATE

D3D11\_MESSAGE\_ID\_LIVE\_DEPTHSTENCILSTATE\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_DEPTHSTENCILSTATE

D3D11\_MESSAGE\_ID\_CREATE\_RASTERIZERSTATE

D3D11\_MESSAGE\_ID\_LIVE\_RASTERIZERSTATE\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_RASTERIZERSTATE

D3D11\_MESSAGE\_ID\_CREATE\_QUERY

D3D11\_MESSAGE\_ID\_LIVE\_QUERY\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_QUERY

D3D11\_MESSAGE\_ID\_CREATE\_PREDICATE

D3D11\_MESSAGE\_ID\_LIVE\_PREDICATE\_WIN7

D3D11\_MESSAGE\_ID\_DESTROY\_PREDICATE

D3D11\_MESSAGE\_ID\_CREATE\_COUNTER

D3D11\_MESSAGE\_ID\_DESTROY\_COUNTER

D3D11\_MESSAGE\_ID\_CREATE\_COMMANDLIST

D3D11\_MESSAGE\_ID\_LIVE\_COMMANDLIST

D3D11\_MESSAGE\_ID\_DESTROY\_COMMANDLIST

D3D11\_MESSAGE\_ID\_CREATE\_CLASSINSTANCE

D3D11\_MESSAGE\_ID\_LIVE\_CLASSINSTANCE

D3D11\_MESSAGE\_ID\_DESTROY\_CLASSINSTANCE

D3D11\_MESSAGE\_ID\_CREATE\_CLASSLINKAGE

D3D11\_MESSAGE\_ID\_LIVE\_CLASSLINKAGE

D3D11\_MESSAGE\_ID\_DESTROY\_CLASSLINKAGE

D3D11\_MESSAGE\_ID\_LIVE\_DEVICE\_WIN7

D3D11\_MESSAGE\_ID\_LIVE\_OBJECT\_SUMMARY\_WIN7

D3D11\_MESSAGE\_ID\_CREATE\_COMPUTESHADER

D3D11\_MESSAGE\_ID\_LIVE\_COMPUTESHADER

D3D11\_MESSAGE\_ID\_DESTROY\_COMPUTESHADER

D3D11\_MESSAGE\_ID\_CREATE\_UNORDEREDACCESSVIEW

D3D11\_MESSAGE\_ID\_LIVE\_UNORDEREDACCESSVIEW

D3D11\_MESSAGE\_ID\_DESTROY\_UNORDEREDACCESSVIEW

D3D11\_MESSAGE\_ID\_DEVICE\_SETSHADER\_INTERFACES\_FEATURELEVEL

D3D11\_MESSAGE\_ID\_DEVICE\_SETSHADER\_INTERFACE\_COUNT\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_SETSHADER\_INVALID\_INSTANCE

D3D11\_MESSAGE\_ID\_DEVICE\_SETSHADER\_INVALID\_INSTANCE\_INDEX

D3D11\_MESSAGE\_ID\_DEVICE\_SETSHADER\_INVALID\_INSTANCE\_TYPE

D3D11\_MESSAGE\_ID\_DEVICE\_SETSHADER\_INVALID\_INSTANCE\_DATA

D3D11\_MESSAGE\_ID\_DEVICE\_SETSHADER\_UNBOUND\_INSTANCE\_DATA

D3D11\_MESSAGE\_ID\_DEVICE\_SETSHADER\_INSTANCE\_DATA\_BINDINGS

D3D11\_MESSAGE\_ID\_DEVICE\_CREATESHADER\_CLASSLINKAGE\_FULL

D3D11\_MESSAGE\_ID\_DEVICE\_CHECKFEATURESUPPORT\_UNRECOGNIZED\_FEATURE

D3D11\_MESSAGE\_ID\_DEVICE\_CHECKFEATURESUPPORT\_MISMATCHED\_DATA\_SIZE

D3D11\_MESSAGE\_ID\_DEVICE\_CHECKFEATURESUPPORT\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETSHADERRESOURCES\_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETCONSTANTBUFFERS\_HAZARD

D3D11\_MESSAGE\_ID\_CSSETSHADERRESOURCES\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_CSSETCONSTANTBUFFERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_CREATECOMPUTESHADER\_INVALIDCALL

D3D11\_MESSAGE\_ID\_CREATECOMPUTESHADER\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_CREATECOMPUTESHADER\_INVALIDSHADERBYTECODE

D3D11\_MESSAGE\_ID\_CREATECOMPUTESHADER\_INVALIDSHADERTYPE

D3D11\_MESSAGE\_ID\_CREATECOMPUTESHADER\_INVALIDCLASSLINKAGE

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_CSSETCONSTANTBUFFERS\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_CSGETSHADERRESOURCES\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_CSGETCONSTANTBUFFERS\_BUFFERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_CSGETSAMPLERS\_SAMPLERS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEVERTEXSHADER\_DOUBLEFLOATOPSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEHULLSHADER\_DOUBLEFLOATOPSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEDOMAINSHADER\_DOUBLEFLOATOPSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADER\_DOUBLEFLOATOPSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_DOUBLEFLOATOPSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEPIXELSHADER\_DOUBLEFLOATOPSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATECOMPUTESHADER\_DOUBLEFLOATOPSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_CREATEBUFFER\_INVALIDSTRUCTURESTRIDE

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDRESOURCE

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDDESC

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDDIMENSIONS

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_UNRECOGNIZEDFORMAT

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS\_HAZARD

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS\_OVERLAPPING\_OLD\_SLOTS

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS\_NO\_OP

D3D11\_MESSAGE\_ID\_CSSETUNORDEREDACCESSVIEWS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_PSSETUNORDEREDACCESSVIEWS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETUNORDEREDACCESSVIEWS\_HAZARD

D3D11\_MESSAGE\_ID\_CLEARUNORDEREDACCESSVIEW\_DENORMFLUSH

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETUNORDEREDACCESSS\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_DEVICE\_CSGETUNORDEREDACCESSS\_VIEWS\_EMPTY

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCVIEW\_TOOMANYOBJECTS

D3D11\_MESSAGE\_ID\_DEVICE\_DISPATCHINDIRECT\_INVALID\_ARG\_BUFFER

D3D11\_MESSAGE\_ID\_DEVICE\_DISPATCHINDIRECT\_OFFSET\_UNALIGNED

D3D11\_MESSAGE\_ID\_DEVICE\_DISPATCHINDIRECT\_OFFSET\_OVERFLOW

D3D11\_MESSAGE\_ID\_DEVICE\_SETRESOURCENLOD\_INVALIDCONTEXT

D3D11\_MESSAGE\_ID\_DEVICE\_SETRESOURCENLOD\_INVALIDRESOURCE

D3D11\_MESSAGE\_ID\_DEVICE\_SETRESOURCENLOD\_INVALIDNLOD

D3D11\_MESSAGE\_ID\_DEVICE\_GETRESOURCENLOD\_INVALIDCONTEXT

D3D11\_MESSAGE\_ID\_DEVICE\_GETRESOURCENLOD\_INVALIDRESOURCE

D3D11\_MESSAGE\_ID\_OMSETDEPTHSTENCIL\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_CLEARDEPTHSTENCILVIEW\_DEPTH\_READONLY

D3D11\_MESSAGE\_ID\_CLEARDEPTHSTENCILVIEW\_STENCIL\_READONLY

D3D11\_MESSAGE\_ID\_CHECKFEATURESUPPORT\_FORMAT\_DEPRECATED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_RETURN\_TYPE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_UNORDEREDACCESSVIEW\_RENDERTARGETVIEW\_OVERLAP

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_DIMENSION\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_APPEND\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_ATOMICS\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_STRUCTURE\_STRIDE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_BUFFER\_TYPE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_RAW\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_FORMAT\_LD\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_FORMAT\_STORE\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_ATOMIC\_ADD\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_ATOMIC\_BITWISE\_OPS\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_ATOMIC\_CMPSTORE\_CMPEXCHANGE\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_ATOMIC\_EXCHANGE\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_ATOMIC\_SIGNED\_MINMAX\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_ATOMIC\_UNSIGNED\_MINMAX\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DISPATCH\_BOUND\_RESOURCE\_MAPPED

D3D11\_MESSAGE\_ID\_DEVICE\_DISPATCH\_THREADGROUPCOUNT\_OVERFLOW

D3D11\_MESSAGE\_ID\_DEVICE\_DISPATCH\_THREADGROUPCOUNT\_ZERO

D3D11\_MESSAGE\_ID\_DEVICE\_SHADERRESOURCEVIEW\_STRUCTURE\_STRIDE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_SHADERRESOURCEVIEW\_BUFFER\_TYPE\_MISMATCH

D3D11\_MESSAGE\_ID\_DEVICE\_SHADERRESOURCEVIEW\_RAW\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DISPATCH\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_DISPATCHINDIRECT\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_COPYSTRUCTURECOUNT\_INVALIDOFFSET

D3D11\_MESSAGE\_ID\_COPYSTRUCTURECOUNT\_LARGEOFFSET

D3D11\_MESSAGE\_ID\_COPYSTRUCTURECOUNT\_INVALIDDESTINATIONSTATE

D3D11\_MESSAGE\_ID\_COPYSTRUCTURECOUNT\_INVALIDSOURCESTATE

D3D11\_MESSAGE\_ID\_CHECKFORMATSUPPORT\_FORMAT\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETUNORDEREDACCESSVIEWS\_INVALIDVIEW

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETUNORDEREDACCESSVIEWS\_INVALIDOFFSET

D3D11\_MESSAGE\_ID\_DEVICE\_CSSETUNORDEREDACCESSVIEWS\_TOOMANYVIEWS

D3D11\_MESSAGE\_ID\_CLEARUNORDEREDACCESSVIEWFLOAT\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_COUNTER\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_REF\_WARNING

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_PIXEL\_SHADER\_WITHOUT\_RTV\_OR\_DSV

D3D11\_MESSAGE\_ID\_SHADER\_ABORT

D3D11\_MESSAGE\_ID\_SHADER\_MESSAGE

D3D11\_MESSAGE\_ID\_SHADER\_ERROR

D3D11\_MESSAGE\_ID\_OFFERRESOURCES\_INVALIDRESOURCE

D3D11\_MESSAGE\_ID\_HSSETSAMPLERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_DSSETSAMPLERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_CSSETSAMPLERS\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_HSSETSHADER\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_DSSETSHADER\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_CSSETSHADER\_UNBINDELETEINGOBJECT

D3D11\_MESSAGE\_ID\_ENQUEUESETEVENT\_INVALIDARG\_RETURN

D3D11\_MESSAGE\_ID\_ENQUEUESETEVENT\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_ENQUEUESETEVENT\_ACCESSDENIED\_RETURN

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS\_NUMUAVS\_INVALIDRANGE

D3D11\_MESSAGE\_ID\_USE\_OF\_ZERO\_REFCOUNT\_OBJECT

D3D11\_MESSAGE\_ID\_D3D11\_MESSAGES\_END

D3D11\_MESSAGE\_ID\_D3D11\_1\_MESSAGES\_START

Value: 0x300000

D3D11\_MESSAGE\_ID\_CREATE\_VIDEODECODER

D3D11\_MESSAGE\_ID\_CREATE\_VIDEOPROCESSORENUM

D3D11\_MESSAGE\_ID\_CREATE\_VIDEOPROCESSOR

D3D11\_MESSAGE\_ID\_CREATE\_DECODEROUTPUTVIEW

D3D11\_MESSAGE\_ID\_CREATE\_PROCESSORINPUTVIEW

D3D11\_MESSAGE\_ID\_CREATE\_PROCESSOROUTPUTVIEW

D3D11\_MESSAGE\_ID\_CREATE\_DEVICECONTEXTSTATE

D3D11\_MESSAGE\_ID\_LIVE\_VIDEODECODER

D3D11\_MESSAGE\_ID\_LIVE\_VIDEOPROCESSORENUM

D3D11\_MESSAGE\_ID\_LIVE\_VIDEOPROCESSOR

D3D11\_MESSAGE\_ID\_LIVE\_DECODEROUTPUTVIEW

D3D11\_MESSAGE\_ID\_LIVE\_PROCESSORINPUTVIEW

D3D11\_MESSAGE\_ID\_LIVE\_PROCESSOROUTPUTVIEW

D3D11\_MESSAGE\_ID\_LIVE\_DEVICECONTEXTSTATE

D3D11\_MESSAGE\_ID\_DESTROY\_VIDEODECODER

D3D11\_MESSAGE\_ID\_DESTROY\_VIDEOPROCESSORENUM

D3D11\_MESSAGE\_ID\_DESTROY\_VIDEOPROCESSOR

D3D11\_MESSAGE\_ID\_DESTROY\_DECODEROUTPUTVIEW

D3D11\_MESSAGE\_ID\_DESTROY\_PROCESSORINPUTVIEW

D3D11\_MESSAGE\_ID\_DESTROY\_PROCESSOROUTPUTVIEW

D3D11\_MESSAGE\_ID\_DESTROY\_DEVICECONTEXTSTATE

D3D11\_MESSAGE\_ID\_CREATEDeviceContextState\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_CREATEDeviceContextState\_INVALIDFEATURELEVEL

D3D11\_MESSAGE\_ID\_CREATEDeviceContextState\_FEATURELEVELS\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_CREATEDeviceContextState\_INVALIDREFIID

D3D11\_MESSAGE\_ID\_DEVICE\_DISCARDVIEW\_INVALIDVIEW

D3D11\_MESSAGE\_ID\_COPYSUBSOURCEREGION1\_INVALIDCOPYFLAGS

D3D11\_MESSAGE\_ID\_UPDATESUBRESOURCE1\_INVALIDCOPYFLAGS

D3D11\_MESSAGE\_ID\_CREATEASTERIZERSTATE\_INVALIDFORCEDSAMPLECOUNT

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODER\_OUTOFGMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODER\_NULLPARAM

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODER\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODER\_ZEROWIDTHHEIGHT

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODER\_DRIVER\_INVALIDBUFFERSIZE

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODER\_DRIVER\_INVALIDBUFFERUSAGE

D3D11\_MESSAGE\_ID\_GETVIDEODECODERPROFILECOUNT\_OUTOFGMEMORY

D3D11\_MESSAGE\_ID\_GETVIDEODECODERPROFILE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEODECODERPROFILE\_INVALIDINDEX

D3D11\_MESSAGE\_ID\_GETVIDEODECODERPROFILE\_OUTOFGMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CHECKVIDEODECODERFORMAT\_NULLPARAM

D3D11\_MESSAGE\_ID\_CHECKVIDEODECODERFORMAT\_OUTOFGMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_GETVIDEODECODERCONFIGCOUNT\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEODECODERCONFIGCOUNT\_OUTOFGMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_GETVIDEODECODERCONFIG\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEODECODERCONFIG\_INVALIDINDEX

D3D11\_MESSAGE\_ID\_GETVIDEODECODERCONFIG\_OUTOFGMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_GETDECODERCREATIONPARAMS\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETDECODERDRIVERHANDLE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETDECODERBUFFER\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETDECODERBUFFER\_INVALIDBUFFER

D3D11\_MESSAGE\_ID\_GETDECODERBUFFER\_INVALIDTYPE

D3D11\_MESSAGE\_ID\_GETDECODERBUFFER\_LOCKED

D3D11\_MESSAGE\_ID\_RELEASEDECODERBUFFER\_NULLPARAM

D3D11\_MESSAGE\_ID\_RELEASEDECODERBUFFER\_INVALIDTYPE

D3D11\_MESSAGE\_ID\_RELEASEDECODERBUFFER\_NOTLOCKED

D3D11\_MESSAGE\_ID\_DECODERBEGINFRAME\_NULLPARAM

D3D11\_MESSAGE\_ID\_DECODERBEGINFRAME\_HAZARD

D3D11\_MESSAGE\_ID\_DECODERENDFRAME\_NULLPARAM

D3D11\_MESSAGE\_ID\_SUBMITDECODERBUFFERS\_NULLPARAM

D3D11\_MESSAGE\_ID\_SUBMITDECODERBUFFERS\_INVALIDTYPE

D3D11\_MESSAGE\_ID\_DECODEREXTENSION\_NULLPARAM

D3D11\_MESSAGE\_ID\_DECODEREXTENSION\_INVALIDRESOURCE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORENUMERATOR\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORENUMERATOR\_NULLPARAM

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORENUMERATOR\_INVALIDFRAMEFORMAT

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORENUMERATOR\_INVALIDUSAGE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORENUMERATOR\_INVALIDINPUTFRAMERATE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORENUMERATOR\_INVALIDOUTPUTFRAMERATE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORENUMERATOR\_INVALIDWIDTHHEIGHT

D3D11\_MESSAGE\_ID\_GETVIDEOPROCESSORCONTENTDESC\_NULLPARAM

D3D11\_MESSAGE\_ID\_CHECKVIDEOPROCESSORFORMAT\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEOPROCESSORCAPS\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEOPROCESSORRATECONVERSIONCAPS\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEOPROCESSORRATECONVERSIONCAPS\_INVALIDINDEX

D3D11\_MESSAGE\_ID\_GETVIDEOPROCESSORCUSTOMRATE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEOPROCESSORCUSTOMRATE\_INVALIDINDEX

D3D11\_MESSAGE\_ID\_GETVIDEOPROCESSORFILTERRANGE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEOPROCESSORFILTERRANGE\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOR\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOR\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTTARGETRECT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTBACKGROUNDCOLOR\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTBACKGROUNDCOLOR\_INVALIDALPHA

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTCOLORSPACE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTALPHAFILLMODE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTALPHAFILLMODE\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTALPHAFILLMODE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTALPHAFILLMODE\_INVALIDFILLMODE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTCONSTRICKTION\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTSTEREOMODE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTSTEREOMODE\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTTEXTENSION\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTTARGETRECT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTBACKGROUNDCOLOR\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTCOLORSPACE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTALPHAFILLMODE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTCONSTRICKTION\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTCONSTRICKTION\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTCONSTRICKTION\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTSTEREOMODE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTEXTENSION\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFRAMEFORMAT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFRAMEFORMAT\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFRAMEFORMAT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMCOLORSPACE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMCOLORSPACE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMOUTPUTRATE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMOUTPUTRATE\_INVALIDRATE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMOUTPUTRATE\_INVALIDFLAG

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMOUTPUTRATE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSOURCERECT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSOURCERECT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSOURCERECT\_INVALIDRECT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMDESTRECT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMDESTRECT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMDESTRECT\_INVALIDRECT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMALPHA\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMALPHA\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMALPHA\_INVALIDALPHA

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPALETTE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPALETTE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPALETTE\_INVALIDCOUNT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPALETTE\_INVALIDALPHA

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPIXELASPECTRATIO\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPIXELASPECTRATIO\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPIXELASPECTRATIO\_INVALIDRATIO

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMLUMAKEY\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMLUMAKEY\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMLUMAKEY\_INVALIDRANGE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMLUMAKEY\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_FLIPUNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_MONOOFFSETUNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_FORMATUNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMSTEREOFORMAT\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMAUTOPROCESSINGMODE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMAUTOPROCESSINGMODE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_INVALIDFILTER

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMFILTER\_INVALIDLEVEL

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMEXTENSION\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMEXTENSION\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMFRAMEFORMAT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMCOLORSPACE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMOUTPUTRATE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMSOURCERECT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMDESTRECT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMALPHA\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMPALETTE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMPIXELASPECTRATIO\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMLUMAKEY\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMSTEREOFORMAT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMAUTOPROCESSINGMODE\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMFILTER\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMEXTENSION\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMEXTENSION\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDSTREAMCOUNT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_TARGETRECT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDOUTPUT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDPASTFRAMES

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDFUTUREFRAMES

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDSOURCERECT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDDESTRECT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDINPUTRESOURCE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDARRAYSIZE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDARRAY

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_RIGHTEXPECTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_RIGHTNOTEXPECTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_STEREONOTENABLED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INVALIDRIGHTRESOURCE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_NOSTEREOSTREAMS

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_INPUTHAZARD

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORBLT\_OUTPUTHAZARD

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_OUTOFGPUMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_NULLPARAM

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_INVALIDTYPE

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_INVALIDBIND

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_UNSUPPORTEDFORMAT

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_INVALIDMIP

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_UNSUPPORTEMIP

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_INVALIDARRAYSIZE

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_INVALIDARRAY

D3D11\_MESSAGE\_ID\_CREATEVIDEODECODEROBJECTVIEW\_INVALIDDIMENSION

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_OUTOFGPUMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_NULLPARAM

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDTYPE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDBIND

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDMISC

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDUSAGE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDFOURCC

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDMIP

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_UNSUPPORTEDMIP

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDARRAYSIZE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDARRAY

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDDIMENSION

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_OUTOFGMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_NULLPARAM

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDTYPE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDBIND

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDFORMAT

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDMIP

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_UNSUPPORTEDMIP

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_UNSUPPORTEDARRAY

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDARRAY

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDDIMENSION

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INVALID\_USE\_OF\_FORCED\_SAMPLE\_COUNT

D3D11\_MESSAGE\_ID\_CREATEBLENDSTATE\_INVALIDLOGICOPS

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDDARRAYWITHDECODER

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDDARRAYWITHDECODER

D3D11\_MESSAGE\_ID\_CREATERENDERTARGETVIEW\_INVALIDDARRAYWITHDECODER

D3D11\_MESSAGE\_ID\_DEVICE\_LOCKEDOUT\_INTERFACE

D3D11\_MESSAGE\_ID\_REF\_WARNING\_ATOMIC\_INCONSISTENT

D3D11\_MESSAGE\_ID\_REF\_WARNING\_READING\_UNINITIALIZED\_RESOURCE

D3D11\_MESSAGE\_ID\_REF\_WARNING\_RAW\_HAZARD

D3D11\_MESSAGE\_ID\_REF\_WARNING\_WAR\_HAZARD

D3D11\_MESSAGE\_ID\_REF\_WARNING\_WAW\_HAZARD

D3D11\_MESSAGE\_ID\_CREATECRYPTOSESSION\_NULLPARAM

D3D11\_MESSAGE\_ID\_CREATECRYPTOSESSION\_OUTOFGMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_GETCRYPTOTYPE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETDECODERPROFILE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONCERTIFICATESIZE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONCERTIFICATE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONCERTIFICATE\_WRONGSIZE

D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONHANDLE\_WRONGSIZE

D3D11\_MESSAGE\_ID\_NEGOTIATECRYPTOSESSIONKEYEXCHANGE\_NULLPARAM

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_NULLPARAM

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_WRONGDEVICE

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_DST\_WRONGDEVICE

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_FORMAT\_MISMATCH

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SIZE\_MISMATCH

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_MULTISAMPLED

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_DST\_NOT\_STAGING

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_MAPPED

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_DST\_MAPPED

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_OFFERED

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_DST\_OFFERED

D3D11\_MESSAGE\_ID\_ENCRYPTIONBLT\_SRC\_CONTENT\_UNDEFINED

D3D11\_MESSAGE\_ID\_DECRIPTIONBLT\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DECRIPTIONBLT\_NULLPARAM

D3D11\_MESSAGE\_ID\_DECRIPTIONBLT\_SRC\_WRONGDEVICE

D3D11\_MESSAGE\_ID\_DECRIPTIONBLT\_DST\_WRONGDEVICE

D3D11\_MESSAGE\_ID\_DECRIPTIONBLT\_FORMAT\_MISMATCH

D3D11\_MESSAGE\_ID\_DECRIPTIONBLT\_SIZE\_MISMATCH

D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_MULTISAMPLED

D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_NOT\_STAGING

D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_NOT\_RENDER\_TARGET

D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_MAPPED

D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_MAPPED

D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_OFFERED

D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_DST\_OFFERED

D3D11\_MESSAGE\_ID\_DECRYPTIONBLT\_SRC\_CONTENT\_UNDEFINED

D3D11\_MESSAGE\_ID\_STARTSESSIONKEYREFRESH\_NULLPARAM

D3D11\_MESSAGE\_ID\_STARTSESSIONKEYREFRESH\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_FINISHSESSIONKEYREFRESH\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETENCRYPTIONBLTKEY\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETENCRYPTIONBLTKEY\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_GETCONTENTPROTECTIONCAPS\_NULLPARAM

D3D11\_MESSAGE\_ID\_CHECKCRYPTOKEYEXCHANGE\_NULLPARAM

D3D11\_MESSAGE\_ID\_CHECKCRYPTOKEYEXCHANGE\_INVALIDINDEX

D3D11\_MESSAGE\_ID\_CREATEAUTENTICATEDCHANNEL\_NULLPARAM

D3D11\_MESSAGE\_ID\_CREATEAUTENTICATEDCHANNEL\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_CREATEAUTENTICATEDCHANNEL\_INVALIDTYPE

D3D11\_MESSAGE\_ID\_CREATEAUTENTICATEDCHANNEL\_OUTOFMEMORY\_RETURN

D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATE\_SIZE\_INVALIDCHANNEL

D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATE\_SIZE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATE\_INVALIDCHANNEL

D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETAUTENTICATEDCHANNELCERTIFICATE\_WRONGSIZE

D3D11\_MESSAGE\_ID\_NEGOTIAEAUTHENTICATEDCHANNELKEYEXCHANGE\_INVALIDCHANNEL

D3D11\_MESSAGE\_ID\_NEGOTIAEAUTHENTICATEDCHANNELKEYEXCHANGE\_NULLPARAM

D3D11\_MESSAGE\_ID\_QUERYAUTHENTICATEDCHANNEL\_NULLPARAM

D3D11\_MESSAGE\_ID\_QUERYAUTHENTICATEDCHANNEL\_WRONGCHANNEL

D3D11\_MESSAGE\_ID\_QUERYAUTHENTICATEDCHANNEL\_UNSUPPORTEDQUERY

D3D11\_MESSAGE\_ID\_QUERYAUTHENTICATEDCHANNEL\_WRONGSIZE

D3D11\_MESSAGE\_ID\_QUERYAUTHENTICATEDCHANNEL\_INVALIDPROCESSINDEX

D3D11\_MESSAGE\_ID\_CONFIGUREAUTHENTICATEDCHANNEL\_NULLPARAM

D3D11\_MESSAGE\_ID\_CONFIGUREAUTHENTICATEDCHANNEL\_WRONGCHANNEL

D3D11\_MESSAGE\_ID\_CONFIGUREAUTHENTICATEDCHANNEL\_UNSUPPORTEDCONFIGURE

D3D11\_MESSAGE\_ID\_CONFIGUREAUTHENTICATEDCHANNEL\_WRONGSIZE

D3D11\_MESSAGE\_ID\_CONFIGUREAUTHENTICATEDCHANNEL\_INVALIDPROCESSIDTYPE

D3D11\_MESSAGE\_ID\_VSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT

D3D11\_MESSAGE\_ID\_DSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT

D3D11\_MESSAGE\_ID\_HSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT

D3D11\_MESSAGE\_ID\_GSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT

D3D11\_MESSAGE\_ID\_PSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT

D3D11\_MESSAGE\_ID\_CSSETCONSTANTBUFFERS\_INVALIDBUFFEROFFSETORCOUNT

D3D11\_MESSAGE\_ID\_NEGOTIAECRPTOSESSIONKEYEXCHANGE\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_NEGOTIAEAUTHENTICATEDCHANNELKEYEXCHANGE\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_OFFERRESOURCES\_INVALIDPRIORITY

D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONHANDLE\_OUTOFMEMORY

D3D11\_MESSAGE\_ID\_ACQUIREHANDLEFORCAPTURE\_NULLPARAM

D3D11\_MESSAGE\_ID\_ACQUIREHANDLEFORCAPTURE\_INVALIDTYPE

D3D11\_MESSAGE\_ID\_ACQUIREHANDLEFORCAPTURE\_INVALIDBIND

D3D11\_MESSAGE\_ID\_ACQUIREHANDLEFORCAPTURE\_INVALIDARRAY

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMROTATION\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMROTATION\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMROTATION\_INVALID

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMROTATION\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMROTATION\_NULLPARAM

D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_INVALIDVIEW

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEVERTEXSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEVERTEXSHADER\_SHADEREXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEHULLSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEHULLSHADER\_SHADEREXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEDOMAINSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEDOMAINSHADER\_SHADEREXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADER\_SHADEREXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_DOUBLEEXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_SHADEREXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEPIXELSHADER\_DOUBLEEXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEPIXELSHADER\_SHADEREXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATECOMPUTESHADER\_DOUBLEEXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATECOMPUTESHADER\_SHADEREXTENSIONSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_SHADER\_LINKAGE\_MINPRECISION

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMALPHA\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMPIXELASPECTRATIO\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEVERTEXSHADER\_UAVSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEHULLSHADER\_UAVSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEDOMAINSHADER\_UAVSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADER\_UAVSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT\_UAVSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATEPIXELSHADER\_UAVSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_CREATECOMPUTESHADER\_UAVSNOTSUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS\_INVALIDOFFSET

D3D11\_MESSAGE\_ID\_DEVICE\_OMSETRENDERTARGETSANDUNORDEREDACCESSVIEWS\_TOOMANYVIEWS

D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_NOTSUPPORTED

D3D11\_MESSAGE\_ID\_SWAPDEVICECONTEXTSTATE\_NOTSUPPORTED

D3D11\_MESSAGE\_ID\_UPDATESUBRESOURCE\_PREFERUPDATESUBRESOURCE1

D3D11\_MESSAGE\_ID\_GETDC\_INACCESSIBLE

D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_INVALIDRECT

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_SAMPLE\_MASK\_IGNORED\_ON\_FL9

D3D11\_MESSAGE\_ID\_DEVICE\_OPEN\_SHARED\_RESOURCE1\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_DEVICE\_OPEN\_SHARED\_RESOURCE\_BY\_NAME\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_ENQUEUESETEVENT\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_OFFERRELEASE\_NOT\_SUPPORTED

D3D11\_MESSAGE\_ID\_OFFERRESOURCES\_INACCESSIBLE

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSORINPUTVIEW\_INVALIDMSAA

D3D11\_MESSAGE\_ID\_CREATEVIDEOPROCESSOROUTPUTVIEW\_INVALIDMSAA

D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_INVALIDSOURCERECT

D3D11\_MESSAGE\_ID\_DEVICE\_CLEARVIEW\_EMPTYRECT

D3D11\_MESSAGE\_ID\_UPDATESUBRESOURCE\_EMPTYDESTBOX

D3D11\_MESSAGE\_ID\_COPYSUBRESOURCEREGION\_EMPTYSOURCEBOX

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_OM\_RENDER\_TARGET\_DOES\_NOT\_SUPPORT\_LOGIC\_OPS

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_DEPTHSTENCILVIEW\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RENDERTARGETVIEW\_NOT\_SET

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_RENDERTARGETVIEW\_NOT\_SET\_DUE\_TO\_FLIP\_PRESENT

D3D11\_MESSAGE\_ID\_DEVICE\_UNORDEREDACCESSVIEW\_NOT\_SET\_DUE\_TO\_FLIP\_PRESENT

D3D11\_MESSAGE\_ID\_GETDATAFORNEWHARDWAREKEY\_NULLPARAM

D3D11\_MESSAGE\_ID\_CHECKCRYPTOSESSIONSTATUS\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETCRYPTOSESSIONPRIVATEDATASIZE\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEODECODERCAPS\_NULLPARAM

D3D11\_MESSAGE\_ID\_GETVIDEODECODERCAPS\_ZEROWIDTHHEIGHT

D3D11\_MESSAGE\_ID\_CHECKVIDEODECODERDOWNSAMPLING\_NULLPARAM

D3D11\_MESSAGE\_ID\_CHECKVIDEODECODERDOWNSAMPLING\_INVALIDCOLORSPACE

D3D11\_MESSAGE\_ID\_CHECKVIDEODECODERDOWNSAMPLING\_ZEROWIDTHHEIGHT

D3D11\_MESSAGE\_ID\_VIDEODECODERENABLEDOWNSAMPLING\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEODECODERENABLEDOWNSAMPLING\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_VIDEODECODERUPDATEDOWNSAMPLING\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEODECODERUPDATEDOWNSAMPLING\_UNSUPPORTED

D3D11\_MESSAGE\_ID\_CHECKVIDEOPROCESSORFORMATCONVERSION\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTCOLORSPACE1\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTCOLORSPACE1\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMCOLORSPACE1\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMCOLORSPACE1\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMMIRROR\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMMIRROR\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMMIRROR\_UNSUPPORTED

D3D11_MESSAGE_ID_VIDEOPROCESSORGETSTREAMCOLORSPACE1_NULLPARAM
D3D11_MESSAGE_ID_VIDEOPROCESSORGETSTREAMMIRROR_NULLPARAM
D3D11_MESSAGE_ID_RECOMMENDVIDEODECODERDOWNSAMPLING_NULLPARAM
D3D11_MESSAGE_ID_RECOMMENDVIDEODECODERDOWNSAMPLING_INVALIDCOLORSPACE
D3D11_MESSAGE_ID_RECOMMENDVIDEODECODERDOWNSAMPLING_ZEROWIDTHHEIGHT
D3D11_MESSAGE_ID_VIDEOPROCESSORSETOUTPUTSHADERUSAGE_NULLPARAM
D3D11_MESSAGE_ID_VIDEOPROCESSORGETOUTPUTSHADERUSAGE_NULLPARAM
D3D11_MESSAGE_ID_VIDEOPROCESSORGETBEHAVIORHINTS_NULLPARAM
D3D11_MESSAGE_ID_VIDEOPROCESSORGETBEHAVIORHINTS_INVALIDSTREAMCOUNT
D3D11_MESSAGE_ID_VIDEOPROCESSORGETBEHAVIORHINTS_TARGETRECT
D3D11_MESSAGE_ID_VIDEOPROCESSORGETBEHAVIORHINTS_INVALIDSOURCERECT
D3D11_MESSAGE_ID_VIDEOPROCESSORGETBEHAVIORHINTS_INVALIDDESTRECT
D3D11_MESSAGE_ID_GETCRYPTOSESSIONPRIVATEDATASIZE_INVALID_KEY_EXCHANGE_TYPE
D3D11_MESSAGE_ID_D3D11_1_MESSAGES_END
D3D11_MESSAGE_ID_D3D11_2_MESSAGES_START
D3D11_MESSAGE_ID_CREATEBUFFER_INVALIDUSAGE
D3D11_MESSAGE_ID_CREATETEXTURE1D_INVALIDUSAGE
D3D11_MESSAGE_ID_CREATETEXTURE2D_INVALIDUSAGE
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_LEVEL9_STEPRATE_NOT_1
D3D11_MESSAGE_ID_CREATEINPUTLAYOUT_LEVEL9_INSTANCING_NOT_SUPPORTED
D3D11_MESSAGE_ID_UPDATETILEMAPPINGS_INVALID_PARAMETER
D3D11_MESSAGE_ID_COPYTILEMAPPINGS_INVALID_PARAMETER
D3D11_MESSAGE_ID_COPYTILES_INVALID_PARAMETER
D3D11_MESSAGE_ID_UPDATETILES_INVALID_PARAMETER
D3D11_MESSAGE_ID_RESIZETILEPOOL_INVALID_PARAMETER

D3D11\_MESSAGE\_ID\_TILEDRESOURCEBARRIER\_INVALID\_PARAMETER

D3D11\_MESSAGE\_ID\_NULL\_TILE\_MAPPING\_ACCESS\_WARNING

D3D11\_MESSAGE\_ID\_NULL\_TILE\_MAPPING\_ACCESS\_ERROR

D3D11\_MESSAGE\_ID\_DIRTY\_TILE\_MAPPING\_ACCESS

D3D11\_MESSAGE\_ID\_DUPLICATE\_TILE\_MAPPINGS\_IN\_COVERED\_AREA

D3D11\_MESSAGE\_ID\_TILE\_MAPPINGS\_IN\_COVERED\_AREA\_DUPPLICATED\_OUTSIDE

D3D11\_MESSAGE\_ID\_TILE\_MAPPINGS\_SHARED\_BETWEEN\_INCOMPATIBLE\_RESOURCES

D3D11\_MESSAGE\_ID\_TILE\_MAPPINGS\_SHARED\_BETWEEN\_INPUT\_AND\_OUTPUT

D3D11\_MESSAGE\_ID\_CHECKMULTISAMPLEQUALITYLEVELS\_INVALIDFLAGS

D3D11\_MESSAGE\_ID\_GETRESOURCETILING\_NONTILED\_RESOURCE

D3D11\_MESSAGE\_ID\_RESIZETILEPOOL\_SHRINK\_WITH\_MAPPINGS\_STILL\_DEFINED\_PAST\_END

D3D11\_MESSAGE\_ID\_NEED\_TO\_CALL\_TILEDRESOURCEBARRIER

D3D11\_MESSAGE\_ID\_CREATEDevice\_INVALIDARGS

D3D11\_MESSAGE\_ID\_CREATEDevice\_WARNING

D3D11\_MESSAGE\_ID\_CLEARUNORDEREDACCESSVIEWINT\_HAZARD

D3D11\_MESSAGE\_ID\_CLEARUNORDEREDACCESSVIEWFLOAT\_HAZARD

D3D11\_MESSAGE\_ID\_TILED\_RESOURCE\_TIER\_1\_BUFFER\_TEXTURE\_MISMATCH

D3D11\_MESSAGE\_ID\_CREATE\_CRYPTOSESSION

D3D11\_MESSAGE\_ID\_CREATE\_AUTHENTICATEDCHANNEL

D3D11\_MESSAGE\_ID\_LIVE\_CRYPTOSESSION

D3D11\_MESSAGE\_ID\_LIVE\_AUTHENTICATEDCHANNEL

D3D11\_MESSAGE\_ID\_DESTROY\_CRYPTOSESSION

D3D11\_MESSAGE\_ID\_DESTROY\_AUTHENTICATEDCHANNEL

D3D11\_MESSAGE\_ID\_D3D11\_2\_MESSAGES\_END

D3D11\_MESSAGE\_ID\_D3D11\_3\_MESSAGES\_START

D3D11\_MESSAGE\_ID\_CREATEASTERIZERSTATE\_INVALID\_CONSERVATIVERASTERMODE

D3D11\_MESSAGE\_ID\_DEVICE\_DRAW\_INVALID\_SYSTEMVALUE

D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_INVALIDCONTEXTTYPE

D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_DECODENOTSUPPORTED

D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_ENCODENOTSUPPORTED

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDPLANEINDEX

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_INVALIDVIDEOPLANEINDEX

D3D11\_MESSAGE\_ID\_CREATESHADERRESOURCEVIEW\_AMBIGUOUSVIDEOPLANEINDEX

D3D11\_MESSAGE\_ID\_CREATERENDERTARGETVIEW\_INVALIDPLANEINDEX

D3D11\_MESSAGE\_ID\_CREATERENDERTARGETVIEW\_INVALIDVIDEOPLANEINDEX

D3D11\_MESSAGE\_ID\_CREATERENDERTARGETVIEW\_AMBIGUOUSVIDEOPLANEINDEX

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDPLANEINDEX

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_INVALIDVIDEOPLANEINDEX

D3D11\_MESSAGE\_ID\_CREATEUNORDEREDACCESSVIEW\_AMBIGUOUSVIDEOPLANEINDEX

D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDSCANDATAOFFSET

D3D11\_MESSAGE\_ID\_JPEGDECODE\_NOTSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGDECODE\_DIMENSIONSTOOLARGE

D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDCOMPONENTS

D3D11\_MESSAGE\_ID\_JPEGDECODE\_DESTINATIONNOT2D

D3D11\_MESSAGE\_ID\_JPEGDECODE\_TILEDRESOURCESUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGDECODE\_GUARDRECTSUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGDECODE\_FORMATUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDMIPLEVEL

D3D11\_MESSAGE\_ID\_JPEGDECODE\_EMPTYDESTBOX

D3D11\_MESSAGE\_ID\_JPEGDECODE\_DESTBOXNOT2D

D3D11\_MESSAGE\_ID\_JPEGDECODE\_DESTBOXNOTSUB

D3D11\_MESSAGE\_ID\_JPEGDECODE\_DESTBOXESINTERSECT

D3D11\_MESSAGE\_ID\_JPEGDECODE\_XSUBSAMPLEMISMATCH

D3D11\_MESSAGE\_ID\_JPEGDECODE\_YSUBSAMPLEMISMATCH

D3D11\_MESSAGE\_ID\_JPEGDECODE\_XSUBSAMPLEODD

D3D11\_MESSAGE\_ID\_JPEGDECODE\_YSUBSAMPLEODD

D3D11\_MESSAGE\_ID\_JPEGDECODE\_OUTPUTDIMENSIONSTOOLARGE

D3D11\_MESSAGE\_ID\_JPEGDECODE\_NONPOW2SCALEUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGDECODE\_FRACTIONALDOWNSCALETOLARGE

D3D11\_MESSAGE\_ID\_JPEGDECODE\_CHROMASIZEMISMATCH

D3D11\_MESSAGE\_ID\_JPEGDECODE\_LUMACHROMASIZEMISMATCH

D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDNUMDESTINATIONS

D3D11\_MESSAGE\_ID\_JPEGDECODE\_SUBBOXUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGDECODE\_1DESTUNSUPPORTEDFORMAT

D3D11\_MESSAGE\_ID\_JPEGDECODE\_3DESTUNSUPPORTEDFORMAT

D3D11\_MESSAGE\_ID\_JPEGDECODE\_SCALEUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDSOURCESIZE

D3D11\_MESSAGE\_ID\_JPEGDECODE\_INVALIDCOPYFLAGS

D3D11\_MESSAGE\_ID\_JPEGDECODE\_HAZARD

D3D11\_MESSAGE\_ID\_JPEGDECODE\_UNSUPPORTEDSRCBUFFERUSAGE

D3D11\_MESSAGE\_ID\_JPEGDECODE\_UNSUPPORTEDSRCBUFFERMISCFLAGS

D3D11\_MESSAGE\_ID\_JPEGDECODE\_UNSUPPORTEDDDSTTEXTUREUSAGE

D3D11\_MESSAGE\_ID\_JPEGDECODE\_BACKBUFFERNOTSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGDECODE\_UNSUPPRTEDCOPYFLAGS

D3D11\_MESSAGE\_ID\_JPEGENCODE\_NOTSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGENCODE\_INVALIDSCANDATAOFFSET

D3D11\_MESSAGE\_ID\_JPEGENCODE\_INVALIDCOMPONENTS

D3D11\_MESSAGE\_ID\_JPEGENCODE\_SOURCENOT2D

D3D11\_MESSAGE\_ID\_JPEGENCODE\_TILEDRESOURCESUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGENCODE\_GUARDRECTSUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGENCODE\_XSUBSAMPLEMISMATCH

D3D11\_MESSAGE\_ID\_JPEGENCODE\_YSUBSAMPLEMISMATCH

D3D11\_MESSAGE\_ID\_JPEGENCODE\_FORMATUNSUPPORTED

D3D11\_MESSAGE\_ID\_JPEGENCODE\_INVALIDSUBRESOURCE

D3D11\_MESSAGE\_ID\_JPEGENCODE\_INVALIDMIPLEVEL

D3D11\_MESSAGE\_ID\_JPEGENCODE\_DIMENSIONSTOOLARGE

D3D11\_MESSAGE\_ID\_JPEGENCODE\_HAZARD

D3D11\_MESSAGE\_ID\_JPEGENCODE\_UNSUPPORTEDDDSTBUFFERUSAGE

D3D11\_MESSAGE\_ID\_JPEGENCODE\_UNSUPPORTEDDDSTBUFFERMISCFLAGS

D3D11\_MESSAGE\_ID\_JPEGENCODE\_UNSUPPORTEDSRCTEXTUREUSAGE

D3D11\_MESSAGE\_ID\_JPEGENCODE\_BACKBUFFERNOTSUPPORTED

D3D11\_MESSAGE\_ID\_CREATEQUERYORPREDICATE\_UNSUPPORTEDCONTEXTTYPEFORQUERY

D3D11\_MESSAGE\_ID\_FLUSH1\_INVALIDCONTEXTTYPE

D3D11\_MESSAGE\_ID\_DEVICE\_SETHARDWAREPROTECTION\_INVALIDCONTEXT

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTHDRMETADATA\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETOUTPUTHDRMETADATA\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTHDRMETADATA\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETOUTPUTHDRMETADATA\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMHDRMETADATA\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMHDRMETADATA\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORSETSTREAMHDRMETADATA\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMHDRMETADATA\_NULLPARAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMHDRMETADATA\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMHDRMETADATA\_INVALIDSIZE

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMFRAMEFORMAT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMCOLORSPACE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMOUTPUTRATE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMSOURCERECT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMDESTRECT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMALPHA\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMPALETTE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMPIXELASPECTRATIO\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMLUMAKEY\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMSTEREOFORMAT\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMAUTOPROCESSINGMODE\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMFILTER\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMROTATION\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMCOLORSPACE1\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_VIDEOPROCESSORGETSTREAMMIRROR\_INVALIDSTREAM

D3D11\_MESSAGE\_ID\_CREATE\_FENCE

D3D11\_MESSAGE\_ID\_LIVE\_FENCE

D3D11\_MESSAGE\_ID\_DESTROY\_FENCE

D3D11\_MESSAGE\_ID\_CREATE\_SYNCHRONIZEDCHANNEL

D3D11\_MESSAGE\_ID\_LIVE\_SYNCHRONIZEDCHANNEL

D3D11_MESSAGE_ID_DESTROY_SYNCHRONIZEDCHANNEL
D3D11_MESSAGE_ID_CREATEFENCE_INVALIDFLAGS
D3D11_MESSAGE_ID_D3D11_3_MESSAGES_END
D3D11_MESSAGE_ID_D3D11_5_MESSAGES_START
D3D11_MESSAGE_ID_NEGOTIATECRYPTOSESSIONKEYEXCHANGEMT_INVALIDKEYEXCHANGETYPE
D3D11_MESSAGE_ID_NEGOTIATECRYPTOSESSIONKEYEXCHANGEMT_NOT_SUPPORTED
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_COMPONENT_COUNT
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_COMPONENT
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_BUFFER_SIZE
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_BUFFER_USAGE
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_BUFFER_MISC_FLAGS
D3D11_MESSAGE_ID_DECODERBEGINFRAME_INVALID_HISTOGRAM_BUFFER_OFFSET
D3D11_MESSAGE_ID_D3D11_5_MESSAGES_END

## Requirements

Header	d3d11sdklayers.h
--------	------------------

## See also

[Layer Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_MESSAGE\_SEVERITY enumeration (d3d11sdklayers.h)

Article 02/22/2024

Debug message severity levels for an information queue.

## Syntax

C++

```
typedef enum D3D11_MESSAGE_SEVERITY {
    D3D11_MESSAGE_SEVERITY_CORRUPTION = 0,
    D3D11_MESSAGE_SEVERITY_ERROR,
    D3D11_MESSAGE_SEVERITY_WARNING,
    D3D11_MESSAGE_SEVERITY_INFO,
    D3D11_MESSAGE_SEVERITY_MESSAGE
};
```

## Constants

[+] Expand table

D3D11\_MESSAGE\_SEVERITY\_CORRUPTION

Value: 0

Defines some type of corruption which has occurred.

D3D11\_MESSAGE\_SEVERITY\_ERROR

Defines an error message.

D3D11\_MESSAGE\_SEVERITY\_WARNING

Defines a warning message.

D3D11\_MESSAGE\_SEVERITY\_INFO

Defines an information message.

D3D11\_MESSAGE\_SEVERITY\_MESSAGE

Defines a message other than corruption, error, warning, or information.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

# Remarks

Use these values to allow or deny message categories to pass through the storage and retrieval filters for an information queue (see [D3D11\\_INFO\\_QUEUE\\_FILTER](#)). This API is used by [ID3D11InfoQueue::AddApplicationMessage](#).

# Requirements

  [Expand table](#)

Requirement	Value
Header	d3d11sdklayers.h

## See also

[Layer Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_RLDO\_FLAGS enumeration (d3d11sdklayers.h)

Article 01/31/2022

Options for the amount of information to report about a device object's lifetime.

## Syntax

C++

```
typedef enum D3D11_RLDO_FLAGS {
    D3D11_RLDO_SUMMARY = 0x1,
    D3D11_RLDO_DETAIL = 0x2,
    D3D11_RLDO_IGNORE_INTERNAL = 0x4
} ;
```

## Constants

D3D11\_RLDO\_SUMMARY

Value: 0x1

Specifies to obtain a summary about a device object's lifetime.

D3D11\_RLDO\_DETAIL

Value: 0x2

Specifies to obtain detailed information about a device object's lifetime.

D3D11\_RLDO\_IGNORE\_INTERNAL

Value: 0x4

This flag indicates to ignore objects which have no external refcounts keeping them alive. D3D objects are printed using an external refcount and an internal refcount. Typically, all objects are printed. This flag means ignore the objects whose external refcount is 0, because the application is not responsible for keeping them alive.

## Remarks

This enumeration is used by [ID3D11Debug::ReportLiveDeviceObjects](#).

Several inline functions exist to combine the options using operators, see the D3D11SDKLayers.h header file for details.

# Requirements

Header	d3d11sdklayers.h
--------	------------------

## See also

[Core Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SHADER\_TRACKING\_OPTIONS enumeration (d3d11sdklayers.h)

Article 01/31/2022

Options that specify how to perform shader debug tracking.

## Syntax

C++

```
typedef enum D3D11_SHADER_TRACKING_OPTION {
    D3D11_SHADER_TRACKING_OPTION_IGNORE = 0,
    D3D11_SHADER_TRACKING_OPTION_TRACK_UNINITIALIZED = 0x1,
    D3D11_SHADER_TRACKING_OPTION_TRACK_RAW = 0x2,
    D3D11_SHADER_TRACKING_OPTION_TRACK_WAR = 0x4,
    D3D11_SHADER_TRACKING_OPTION_TRACK_WAW = 0x8,
    D3D11_SHADER_TRACKING_OPTION_ALLOW_SAME = 0x10,
    D3D11_SHADER_TRACKING_OPTION_TRACK_ATOMIC_CONSISTENCY = 0x20,
    D3D11_SHADER_TRACKING_OPTION_TRACK_RAW_ACROSS_THREADGROUPS = 0x40,
    D3D11_SHADER_TRACKING_OPTION_TRACK_WAR_ACROSS_THREADGROUPS = 0x80,
    D3D11_SHADER_TRACKING_OPTION_TRACK_WAW_ACROSS_THREADGROUPS = 0x100,
    D3D11_SHADER_TRACKING_OPTION_TRACK_ATOMIC_CONSISTENCY_ACROSS_THREADGROUPS
= 0x200,
    D3D11_SHADER_TRACKING_OPTION_UAV_SPECIFIC_FLAGS,
    D3D11_SHADER_TRACKING_OPTION_ALL_HAZARDS,
    D3D11_SHADER_TRACKING_OPTION_ALL_HAZARDS_ALLOWING_SAME,
    D3D11_SHADER_TRACKING_OPTION_ALL_OPTIONS
} D3D11_SHADER_TRACKING_OPTIONS;
```

## Constants

D3D11\_SHADER\_TRACKING\_OPTION\_IGNORE

Value: 0

No debug tracking is performed.

D3D11\_SHADER\_TRACKING\_OPTION\_TRACK\_UNINITIALIZED

Value: 0x1

Track the reading of uninitialized data.

D3D11\_SHADER\_TRACKING\_OPTION\_TRACK\_RAW

Value: 0x2

Track read-after-write hazards.

`D3D11_SHADER_TRACKING_OPTION_TRACK_WAR`

Value: *0x4*

Track write-after-read hazards.

`D3D11_SHADER_TRACKING_OPTION_TRACK_WAW`

Value: *0x8*

Track write-after-write hazards.

`D3D11_SHADER_TRACKING_OPTION_ALLOW_SAME`

Value: *0x10*

Track that hazards are allowed in which data is written but the value does not change.

`D3D11_SHADER_TRACKING_OPTION_TRACK_ATOMIC_CONSISTENCY`

Value: *0x20*

Track that only one type of atomic operation is used on an address.

`D3D11_SHADER_TRACKING_OPTION_TRACK_RAW_ACROSS_THREADGROUPS`

Value: *0x40*

Track read-after-write hazards across thread groups.

`D3D11_SHADER_TRACKING_OPTION_TRACK_WAR_ACROSS_THREADGROUPS`

Value: *0x80*

Track write-after-read hazards across thread groups.

`D3D11_SHADER_TRACKING_OPTION_TRACK_WAW_ACROSS_THREADGROUPS`

Value: *0x100*

Track write-after-write hazards across thread groups.

`D3D11_SHADER_TRACKING_OPTION_TRACK_ATOMIC_CONSISTENCY_ACROSS_THREADGROUPS`

Value: *0x200*

Track that only one type of atomic operation is used on an address across thread groups.

`D3D11_SHADER_TRACKING_OPTION_UAV_SPECIFIC_FLAGS`

Track hazards that are specific to unordered access views (UAVs).

`D3D11_SHADER_TRACKING_OPTION_ALL_HAZARDS`

Track all hazards.

`D3D11_SHADER_TRACKING_OPTION_ALL_HAZARDS_ALLOWING_SAME`

Track all hazards and track that hazards are allowed in which data is written but the value does not change.

`D3D11_SHADER_TRACKING_OPTION_ALL_OPTIONS`

All of the preceding tracking options are set except

`D3D11_SHADER_TRACKING_OPTION_IGNORE`.

## Remarks

This enumeration is used by the following methods:

- [ID3D11RefDefaultTrackingOptions::SetTrackingOptions](#)
- [ID3D11RefTrackingOptions::SetTrackingOptions](#)
- [ID3D11TracingDevice::SetShaderTrackingOptions](#)
- [ID3D11TracingDevice::SetShaderTrackingOptionsByType](#)

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11sdklayers.h

## See also

[Layer Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SHADER\_TRACKING\_RESOURCE\_TYPE enumeration (d3d11sdklayers.h)

Article 01/31/2022

Indicates which resource types to track.

## Syntax

C++

```
typedef enum D3D11_SHADER_TRACKING_RESOURCE_TYPE {
    D3D11_SHADER_TRACKING_RESOURCE_TYPE_NONE = 0,
    D3D11_SHADER_TRACKING_RESOURCE_TYPE_UAV_DEVICEMEMORY = 1,
    D3D11_SHADER_TRACKING_RESOURCE_TYPE_NON_UAV_DEVICEMEMORY = 2,
    D3D11_SHADER_TRACKING_RESOURCE_TYPE_ALL_DEVICEMEMORY = 3,
    D3D11_SHADER_TRACKING_RESOURCE_TYPE_GROUPSHARED_MEMORY = 4,
    D3D11_SHADER_TRACKING_RESOURCE_TYPE_ALL_SHARED_MEMORY = 5,
    D3D11_SHADER_TRACKING_RESOURCE_TYPE_GROUPSHARED_NON_UAV = 6,
    D3D11_SHADER_TRACKING_RESOURCE_TYPE_ALL = 7
};
```

## Constants

D3D11\_SHADER\_TRACKING\_RESOURCE\_TYPE\_NONE

Value: 0

No resource types are tracked.

D3D11\_SHADER\_TRACKING\_RESOURCE\_TYPE\_UAV\_DEVICEMEMORY

Value: 1

Track device memory that is created with unordered access view (UAV) bind flags.

D3D11\_SHADER\_TRACKING\_RESOURCE\_TYPE\_NON\_UAV\_DEVICEMEMORY

Value: 2

Track device memory that is created without UAV bind flags.

D3D11\_SHADER\_TRACKING\_RESOURCE\_TYPE\_ALL\_DEVICEMEMORY

Value: 3

Track all device memory.

D3D11\_SHADER\_TRACKING\_RESOURCE\_TYPE\_GROUPSHARED\_MEMORY

Value: 4

Track all shaders that use group shared memory.

`D3D11_SHADER_TRACKING_RESOURCE_TYPE_ALL_SHARED_MEMORY`

Value: 5

Track all device memory except device memory that is created without UAV bind flags.

`D3D11_SHADER_TRACKING_RESOURCE_TYPE_GROUPSHARED_NON_UAV`

Value: 6

Track all device memory except device memory that is created with UAV bind flags.

`D3D11_SHADER_TRACKING_RESOURCE_TYPE_ALL`

Value: 7

Track all memory on the device.

## Remarks

The [ID3D11TracingDevice::SetShaderTrackingOptionsByType](#) or [ID3D11RefDefaultTrackingOptions::SetTrackingOptions](#) method tracks a specific type of resource.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

**Minimum supported client** Windows 8 [desktop apps only]

**Minimum supported server** Windows Server 2012 [desktop apps only]

**Header** d3d11sdklayers.h

## See also

[Layer Enumerations](#)

## Feedback

Was this page helpful? [!\[\]\(0568175fad002c54ec231f71115018c0\_img.jpg\) Yes](#) [!\[\]\(b0668c0feea707c4b284cac6f7d972d4\_img.jpg\) No](#)

Get help at Microsoft Q&A

# Resource Reference (Direct3D 11 Graphics)

Article • 12/10/2020

The Direct3D API defines several API elements to help you create and manage resources.

## In this section

Topic	Description
Resource Interfaces	<p>There are a number of interfaces for the two basic types of resources: buffers and textures. There are also interfaces for views of resources.</p> <p>An application uses a view to bind a resource to a pipeline stage. The view defines how the resource can be accessed during rendering. This section describes the view interfaces.</p>
Resource Functions	This section contains information about the resource functions.
Resource Structures	Structures are used to create and use resources.
Resource Enumerations	Enumerations are used to specify information about how resources are created and accessed during rendering.

## Related topics

[Direct3D 11 Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Resource Interfaces (Direct3D 11 Graphics)

Article • 12/10/2020

There are a number of interfaces for the two basic types of resources: buffers and textures. There are also interfaces for views of resources.

An application uses a view to bind a resource to a pipeline stage. The view defines how the resource can be accessed during rendering. This section describes the view interfaces.

## In this section

Topic	Description
<a href="#">ID3D11Buffer</a>	A buffer interface accesses a buffer resource, which is unstructured memory. Buffers typically store vertex or index data.
<a href="#">ID3D11DepthStencilView</a>	A depth-stencil-view interface accesses a texture resource during depth-stencil testing.
<a href="#">ID3D11RenderTargetView</a>	A render-target-view interface identifies the render-target subresources that can be accessed during rendering.
<a href="#">ID3D11RenderTargetView1</a>	A render-target-view interface represents the render-target subresources that can be accessed during rendering.
<a href="#">ID3D11Resource</a>	A resource interface provides common actions on all resources.
<a href="#">ID3D11ShaderResourceView</a>	A shader-resource-view interface specifies the subresources a shader can access during rendering. Examples of shader resources include a constant buffer, a texture buffer, and a texture.
<a href="#">ID3D11ShaderResourceView1</a>	A shader-resource-view interface represents the subresources a shader can access during rendering. Examples of shader resources include a constant buffer, a texture buffer, and a texture.
<a href="#">ID3D11Texture1D</a>	A 1D texture interface accesses texel data, which is structured memory.
<a href="#">ID3D11Texture2D</a>	A 2D texture interface manages texel data, which is structured memory.

Topic	Description
<a href="#">ID3D11Texture2D1</a>	A 2D texture interface represents texel data, which is structured memory.
<a href="#">ID3D11Texture3D</a>	A 3D texture interface accesses texel data, which is structured memory.
<a href="#">ID3D11Texture3D1</a>	A 3D texture interface represents texel data, which is structured memory.
<a href="#">ID3D11UnorderedAccessView</a>	A view interface specifies the parts of a resource the pipeline can access during rendering.
<a href="#">ID3D11UnorderedAccessView1</a>	An unordered-access-view interface represents the parts of a resource the pipeline can access during rendering.
<a href="#">ID3D11View</a>	A view interface specifies the parts of a resource the pipeline can access during rendering.

## Related topics

[Resource Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Buffer interface (d3d11.h)

Article 02/22/2024

A buffer interface accesses a buffer resource, which is unstructured memory. Buffers typically store vertex or index data.

## Inheritance

The **ID3D11Buffer** interface inherits from [ID3D11Resource](#). **ID3D11Buffer** also has these types of members:

## Methods

The **ID3D11Buffer** interface has these methods.

[+] Expand table

### [ID3D11Buffer::GetDesc](#)

Get the properties of a buffer resource. ([ID3D11Buffer::GetDesc](#))

## Remarks

There are three types of buffers: vertex, index, or a shader-constant buffer. Create a buffer resource by calling [ID3D11Device::CreateBuffer](#).

A buffer must be bound to the pipeline before it can be accessed. Buffers can be bound to the input-assembler stage by calls to [ID3D11DeviceContext::IASetVertexBuffers](#) and [ID3D11DeviceContext::IASetIndexBuffer](#), to the stream-output stage by a call to [ID3D11DeviceContext::SOSetTargets](#), and to a shader stage by calling the appropriate shader method (such as [ID3D11DeviceContext::VSSetConstantBuffers](#) for example).

Buffers can be bound to multiple pipeline stages simultaneously for reading. A buffer can also be bound to a single pipeline stage for writing; however, the same buffer cannot be bound for reading and writing simultaneously.

## Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11Resource](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Buffer::GetDesc method (d3d11.h)

Article 02/22/2024

Get the properties of a buffer resource.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_BUFFER_DESC *pDesc  
) ;
```

## Parameters

[out] pDesc

Type: [D3D11\\_BUFFER\\_DESC\\*](#)

Pointer to a resource description (see [D3D11\\_BUFFER\\_DESC](#)) filled in by the method.

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Buffer](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DepthStencilView interface (d3d11.h)

Article 02/22/2024

A depth-stencil-view interface accesses a texture resource during depth-stencil testing.

## Inheritance

The **ID3D11DepthStencilView** interface inherits from [ID3D11View](#).

**ID3D11DepthStencilView** also has these types of members:

## Methods

The **ID3D11DepthStencilView** interface has these methods.

[+] Expand table

<p><a href="#">ID3D11DepthStencilView::GetDesc</a></p> <p>Get the depth-stencil view. (<code>ID3D11DepthStencilView.GetDesc</code>)</p>

## Remarks

To create a depth-stencil view, call [ID3D11Device::CreateDepthStencilView](#).

To bind a depth-stencil view to the pipeline, call [ID3D11DeviceContext::OMSetRenderTargets](#).

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11View](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?



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

# ID3D11DepthStencilView::GetDesc method (d3d11.h)

Article 02/22/2024

Get the depth-stencil view.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_DEPTH_STENCIL_VIEW_DESC *pDesc  
) ;
```

## Parameters

[out] pDesc

Type: [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC\\*](#)

Pointer to a depth-stencil-view description (see [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)).

## Return value

None

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RenderTargetView interface (d3d11.h)

Article 02/22/2024

A render-target-view interface identifies the render-target subresources that can be accessed during rendering.

## Inheritance

The [ID3D11RenderTargetView](#) interface inherits from [ID3D11View](#).

[ID3D11RenderTargetView](#) also has these types of members:

## Methods

The [ID3D11RenderTargetView](#) interface has these methods.

[+] Expand table

### [ID3D11RenderTargetView::GetDesc](#)

Get the properties of a render target view. ([ID3D11RenderTargetView::GetDesc](#))

## Remarks

To create a render-target view, call [ID3D11Device::CreateRenderTargetView](#). To bind a render-target view to the pipeline, call [ID3D11DeviceContext::OMSetRenderTargets](#).

A rendertarget is a resource that can be written by the output-merger stage at the end of a render pass. Each render-target should also have a corresponding depth-stencil view.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11View](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RenderTargetView::GetDesc method (d3d11.h)

Article 02/22/2024

Get the properties of a render target view.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_RENDER_TARGET_VIEW_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC\\*](#)

Pointer to the description of a render target view (see [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)).

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11RenderTargetView](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RenderTargetView1 interface (d3d11\_3.h)

Article 07/22/2021

A render-target-view interface represents the render-target subresources that can be accessed during rendering.

## Inheritance

The [ID3D11RenderTargetView1](#) interface inherits from [ID3D11RenderTargetView](#).

[ID3D11RenderTargetView1](#) also has these types of members:

## Methods

The [ID3D11RenderTargetView1](#) interface has these methods.

### [ID3D11RenderTargetView1::GetDesc1](#)

Gets the properties of a render-target view.

## Remarks

To create a render-target view, call [ID3D11Device3::CreateRenderTargetView1](#). To bind a render-target view to the pipeline, call [ID3D11DeviceContext::OMSetRenderTargets](#).

A render target is a resource that can be written by the output-merger stage at the end of a render pass. Each render target can also have a corresponding depth-stencil view.

## Requirements

Minimum supported client	Windows 10 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps   UWP apps]
Target Platform	Windows

## See also

[ID3D11RenderTargetView](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11RenderTargetView1::GetDesc1 method (d3d11\_3.h)

Article 02/22/2024

Gets the properties of a render-target view.

## Syntax

C++

```
void GetDesc1(  
    [out] D3D11_RENDER_TARGET_VIEW_DESC1 *pDesc1  
);
```

## Parameters

[out] pDesc1

Type: [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC1\\*](#)

A pointer to a [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC1](#) structure that receives the description of the render-target view.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11RenderTargetView1](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# ID3D11Resource interface (d3d11.h)

Article 02/22/2024

A resource interface provides common actions on all resources.

## Inheritance

The [ID3D11Resource](#) interface inherits from [ID3D11DeviceChild](#). [ID3D11Resource](#) also has these types of members:

## Methods

The [ID3D11Resource](#) interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11Resource::GetEvictionPriority</a>
Get the eviction priority of a resource. ( <a href="#">ID3D11Resource.GetEvictionPriority</a> )
<a href="#">ID3D11Resource::GetType</a>
Get the type of the resource. ( <a href="#">ID3D11Resource.GetType</a> )
<a href="#">ID3D11Resource::SetEvictionPriority</a>
Set the eviction priority of a resource. ( <a href="#">ID3D11Resource.SetEvictionPriority</a> )

## Remarks

You don't directly create a resource interface; instead, you create buffers and textures that inherit from a resource interface. For more info, see [How to: Create a Vertex Buffer](#), [How to: Create an Index Buffer](#), [How to: Create a Constant Buffer](#), and [How to: Create a Texture](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11DeviceChild](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Resource::GetEvictionPriority method (d3d11.h)

Article 02/22/2024

Get the eviction priority of a resource.

## Syntax

C++

```
UINT GetEvictionPriority();
```

## Return value

Type: [UINT](#)

One of the following values, which specifies the eviction priority for the resource:

- DXGI\_RESOURCE\_PRIORITY\_MINIMUM
- DXGI\_RESOURCE\_PRIORITY\_LOW
- DXGI\_RESOURCE\_PRIORITY\_NORMAL
- DXGI\_RESOURCE\_PRIORITY\_HIGH
- DXGI\_RESOURCE\_PRIORITY\_MAXIMUM

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Resource](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Resource::GetType method (d3d11.h)

Article 02/22/2024

Get the type of the resource.

## Syntax

C++

```
void GetType(  
    [out] D3D11_RESOURCE_DIMENSION *pResourceDimension  
);
```

## Parameters

[out] pResourceDimension

Type: [D3D11\\_RESOURCE\\_DIMENSION\\*](#)

Pointer to the resource type (see [D3D11\\_RESOURCE\\_DIMENSION](#)).

## Return value

None

## Remarks

**Windows Phone 8:** This API is supported.

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11Resource](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Resource::SetEvictionPriority method (d3d11.h)

Article 07/27/2022

Set the eviction priority of a resource.

## Syntax

C++

```
void SetEvictionPriority(  
    [in] UINT EvictionPriority  
);
```

## Parameters

[in] `EvictionPriority`

Type: [UINT](#)

Eviction priority for the resource, which is one of the following values:

- `DXGI_RESOURCE_PRIORITY_MINIMUM`
- `DXGI_RESOURCE_PRIORITY_LOW`
- `DXGI_RESOURCE_PRIORITY_NORMAL`
- `DXGI_RESOURCE_PRIORITY_HIGH`
- `DXGI_RESOURCE_PRIORITY_MAXIMUM`

## Return value

None

## Remarks

Resource priorities determine which resource to evict from video memory when the system has run out of video memory. The resource will not be lost; it will be removed from video memory and placed into system memory, or possibly placed onto the hard drive. The resource will be loaded back into video memory when it is required.

A resource that is set to the maximum priority, DXGI\_RESOURCE\_PRIORITY\_MAXIMUM, is only evicted if there is no other way of resolving the incoming memory request. The Windows Display Driver Model (WDDM) tries to split an incoming memory request to its minimum size and evict lower-priority resources before evicting a resource with maximum priority.

Changing the priorities of resources should be done carefully. The wrong eviction priorities could be a detriment to performance rather than an improvement.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11Resource](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ShaderResourceView interface (d3d11.h)

Article 02/22/2024

A shader-resource-view interface specifies the subresources a shader can access during rendering. Examples of shader resources include a constant buffer, a texture buffer, and a texture.

## Inheritance

The [ID3D11ShaderResourceView](#) interface inherits from [ID3D11View](#).

[ID3D11ShaderResourceView](#) also has these types of members:

## Methods

The [ID3D11ShaderResourceView](#) interface has these methods.

[+] Expand table

<a href="#">ID3D11ShaderResourceView::GetDesc</a>
Get the shader resource view's description. ( <a href="#">ID3D11ShaderResourceView::GetDesc</a> )

## Remarks

To create a shader-resource view, call [ID3D11Device::CreateShaderResourceView](#).

A shader-resource view is required when binding a resource to a shader stage; the binding occurs by calling [ID3D11DeviceContext::GSSetShaderResources](#), [ID3D11DeviceContext::VSSetShaderResources](#) or [ID3D11DeviceContext::PSSetShaderResources](#).

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11View](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ShaderResourceView::GetDesc method (d3d11.h)

Article 02/22/2024

Get the shader resource view's description.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_SHADER_RESOURCE_VIEW_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC\\*](#)

A pointer to a [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure to be filled with data about the shader resource view.

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11ShaderResourceView](#)

---

## Feedback

Was this page helpful?



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

# ID3D11ShaderResourceView1 interface (d3d11\_3.h)

Article 02/22/2024

A shader-resource-view interface represents the subresources a shader can access during rendering. Examples of shader resources include a constant buffer, a texture buffer, and a texture.

## Inheritance

The [ID3D11ShaderResourceView1](#) interface inherits from [ID3D11ShaderResourceView](#). [ID3D11ShaderResourceView1](#) also has these types of members:

## Methods

The [ID3D11ShaderResourceView1](#) interface has these methods.

[+] Expand table

<p><a href="#">ID3D11ShaderResourceView1::GetDesc1</a></p> <p>Gets the shader-resource view's description.</p>

## Remarks

To create a shader-resource view, call [ID3D11Device3::CreateShaderResourceView1](#).

A shader-resource view is required when binding a resource to a shader stage; the binding occurs by calling [ID3D11DeviceContext::GSSetShaderResources](#), [ID3D11DeviceContext::VSSetShaderResources](#) or [ID3D11DeviceContext::PSSetShaderResources](#).

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_3.h

## See also

[ID3D11ShaderResourceView](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ShaderResourceView1::GetDesc1 method (d3d11\_3.h)

Article 02/22/2024

Gets the shader-resource view's description.

## Syntax

C++

```
void GetDesc1(  
    [out] D3D11_SHADER_RESOURCE_VIEW_DESC1 *pDesc1  
);
```

## Parameters

[out] pDesc1

Type: [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC1\\*](#)

A pointer to a [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC1](#) structure that receives the description of the shader-resource view.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11ShaderResourceView1](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# ID3D11Texture1D interface (d3d11.h)

Article 02/22/2024

A 1D texture interface accesses texel data, which is structured memory.

## Inheritance

The **ID3D11Texture1D** interface inherits from [ID3D11Resource](#). **ID3D11Texture1D** also has these types of members:

## Methods

The **ID3D11Texture1D** interface has these methods.

[+] [Expand table](#)

### [ID3D11Texture1D::GetDesc](#)

Get the properties of the texture resource. (`ID3D11Texture1D.GetDesc`)

## Remarks

To create an empty 1D texture, call [ID3D11Device::CreateTexture1D](#). For info about how to create a 2D texture, which is similar to creating a 1D texture, see [How to: Create a Texture](#).

Textures cannot be bound directly to the pipeline; instead, a view must be created and bound. Using a view, texture data can be interpreted at run time within certain restrictions. To use the texture as a render target or depth-stencil resource, call [ID3D11Device::CreateRenderTargetView](#), and [ID3D11Device::CreateDepthStencilView](#), respectively. To use the texture as an input to a shader, create a by calling [ID3D11Device::CreateShaderResourceView](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11Resource](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Texture1D::GetDesc method (d3d11.h)

Article 02/22/2024

Get the properties of the texture resource.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_TEXTURE1D_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_TEXTURE1D\\_DESC\\*](#)

Pointer to a resource description (see [D3D11\\_TEXTURE1D\\_DESC](#)).

## Return value

None

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Texture2D interface (d3d11.h)

Article 02/22/2024

A 2D texture interface manages texel data, which is structured memory.

## Inheritance

The **ID3D11Texture2D** interface inherits from [ID3D11Resource](#). **ID3D11Texture2D** also has these types of members:

## Methods

The **ID3D11Texture2D** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11Texture2D::GetDesc</a>
Get the properties of the texture resource. ( <code>ID3D11Texture2D.GetDesc</code> )

## Remarks

To create an empty Texture2D resource, call [ID3D11Device::CreateTexture2D](#). For info about how to create a 2D texture, see [How to: Create a Texture](#).

Textures cannot be bound directly to the pipeline; instead, a view must be created and bound. Using a view, texture data can be interpreted at run time within certain restrictions. To use the texture as a render target or depth-stencil resource, call [ID3D11Device::CreateRenderTargetView](#), and [ID3D11Device::CreateDepthStencilView](#), respectively. To use the texture as an input to a shader, create a by calling [ID3D11Device::CreateShaderResourceView](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11Resource](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Texture2D::GetDesc method (d3d11.h)

Article 02/22/2024

Get the properties of the texture resource.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_TEXTURE2D_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_TEXTURE2D\\_DESC\\*](#)

Pointer to a resource description (see [D3D11\\_TEXTURE2D\\_DESC](#)).

## Return value

None

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Texture2D1 interface (d3d11\_3.h)

Article 02/22/2024

A 2D texture interface represents texel data, which is structured memory.

## Inheritance

The **ID3D11Texture2D1** interface inherits from [ID3D11Texture2D](#). **ID3D11Texture2D1** also has these types of members:

## Methods

The **ID3D11Texture2D1** interface has these methods.

[+] [Expand table](#)

### [ID3D11Texture2D1::GetDesc1](#)

Gets the properties of the texture resource. (**ID3D11Texture2D1::GetDesc1**)

## Remarks

To create an empty Texture2D resource, call [ID3D11Device3::CreateTexture2D1](#). For info about how to create a 2D texture, see [How to: Create a Texture](#).

Textures can't be bound directly to the pipeline; instead, a view must be created and bound. Using a view, texture data can be interpreted at run time within certain restrictions. To use the texture as a render-target or depth-stencil resource, call [ID3D11Device3::CreateRenderTargetView1](#), and [ID3D11Device::CreateDepthStencilView](#), respectively. To use the texture as an input to a shader, call [ID3D11Device3::CreateShaderResourceView1](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_3.h

## See also

[ID3D11Texture2D](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Texture2D1::GetDesc1 method (d3d11\_3.h)

Article 02/22/2024

Gets the properties of the texture resource.

## Syntax

C++

```
void GetDesc1(  
    [out] D3D11_TEXTURE2D_DESC1 *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_TEXTURE2D\\_DESC1\\*](#)

A pointer to a [D3D11\\_TEXTURE2D\\_DESC1](#) structure that receives the description of the 2D texture.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11Texture2D1](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# ID3D11Texture3D interface (d3d11.h)

Article 02/22/2024

A 3D texture interface accesses texel data, which is structured memory.

## Inheritance

The **ID3D11Texture3D** interface inherits from [ID3D11Resource](#). **ID3D11Texture3D** also has these types of members:

## Methods

The **ID3D11Texture3D** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11Texture3D::GetDesc</a>
Get the properties of the texture resource. ( <code>ID3D11Texture3D.GetDesc</code> )

## Remarks

To create an empty Texture3D resource, call [ID3D11Device::CreateTexture3D](#). For info about how to create a 2D texture, which is similar to creating a 3D texture, see [How to: Create a Texture](#).

Textures cannot be bound directly to the pipeline; instead, a view must be created and bound. Using a view, texture data can be interpreted at run time within certain restrictions. To use the texture as a render target or depth-stencil resource, call [ID3D11Device::CreateRenderTargetView](#), and [ID3D11Device::CreateDepthStencilView](#), respectively. To use the texture as an input to a shader, create a by calling [ID3D11Device::CreateShaderResourceView](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11Resource](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Texture3D::GetDesc method (d3d11.h)

Article 02/22/2024

Get the properties of the texture resource.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_TEXTURE3D_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_TEXTURE3D\\_DESC\\*](#)

Pointer to a resource description (see [D3D11\\_TEXTURE3D\\_DESC](#)).

## Return value

None

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11Texture3D1 interface (d3d11\_3.h)

Article07/27/2022

A 3D texture interface represents texel data, which is structured memory.

## Inheritance

The **ID3D11Texture3D1** interface inherits from [ID3D11Texture3D](#). **ID3D11Texture3D1** also has these types of members:

## Methods

The **ID3D11Texture3D1** interface has these methods.

### [ID3D11Texture3D1::GetDesc1](#)

Gets the properties of the texture resource. ([ID3D11Texture3D1.GetDesc1](#))

## Remarks

To create an empty Texture3D resource, call [ID3D11Device3::CreateTexture3D1](#). For info about how to create a 2D texture, which is similar to creating a 3D texture, see [How to: Create a Texture](#).

Textures can't be bound directly to the pipeline; instead, a view must be created and bound. Using a view, texture data can be interpreted at run time within certain restrictions. To use the texture as a render-target or depth-stencil resource, call [ID3D11Device3::CreateRenderTargetView1](#), and [ID3D11Device::CreateDepthStencilView](#), respectively. To use the texture as an input to a shader, call [ID3D11Device3::CreateShaderResourceView1](#).

## Requirements

Minimum supported client	Windows 10 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps   UWP apps]

Target Platform	Windows
Header	d3d11_3.h

## See also

[ID3D11Texture3D](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?



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

# ID3D11Texture3D1::GetDesc1 method (d3d11\_3.h)

Article 02/22/2024

Gets the properties of the texture resource.

## Syntax

C++

```
void GetDesc1(  
    [out] D3D11_TEXTURE3D_DESC1 *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_TEXTURE3D\\_DESC1\\*](#)

A pointer to a [D3D11\\_TEXTURE3D\\_DESC1](#) structure that receives the description of the 3D texture.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11Texture3D1](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11UnorderedAccessView interface (d3d11.h)

Article 02/22/2024

A view interface specifies the parts of a resource the pipeline can access during rendering.

## Inheritance

The [ID3D11UnorderedAccessView](#) interface inherits from [ID3D11View](#).

[ID3D11UnorderedAccessView](#) also has these types of members:

## Methods

The [ID3D11UnorderedAccessView](#) interface has these methods.

[+] Expand table

<p><a href="#">ID3D11UnorderedAccessView::GetDesc</a></p> <p>Get a description of the resource.</p>

## Remarks

To create a view for an unordered access resource, call [ID3D11Device::CreateUnorderedAccessView](#).

All resources must be bound to the pipeline before they can be accessed. Call [ID3D11DeviceContext::CSSetUnorderedAccessViews](#) to bind an unordered access view to a compute shader; call [ID3D11DeviceContext::OMSetRenderTargetsAndUnorderedAccessViews](#) to bind an unordered access view to a pixel shader.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11View](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11UnorderedAccessView::GetDesc method (d3d11.h)

Article 02/22/2024

Get a description of the resource.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_UNORDERED_ACCESS_VIEW_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC\\*](#)

Pointer to a resource description (see [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#).)

## Return value

None

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11UnorderedAccessView1 interface (d3d11\_3.h)

Article 02/22/2024

An unordered-access-view interface represents the parts of a resource the pipeline can access during rendering.

## Inheritance

The **ID3D11UnorderedAccessView1** interface inherits from [ID3D11UnorderedAccessView](#). **ID3D11UnorderedAccessView1** also has these types of members:

## Methods

The **ID3D11UnorderedAccessView1** interface has these methods.

[+] Expand table

<a href="#">ID3D11UnorderedAccessView1::GetDesc1</a>	Gets a description of the resource.

## Remarks

To create a view for an unordered access resource, call [ID3D11Device3::CreateUnorderedAccessView1](#).

All resources must be bound to the pipeline before they can be accessed. Call [ID3D11DeviceContext::CSSetUnorderedAccessViews](#) to bind an unordered access view to a compute shader; call [ID3D11DeviceContext::OMSetRenderTargetAndUnorderedAccessViews](#) to bind an unordered access view to a pixel shader.

## Requirements

Requirement	Value
Minimum supported client	Windows 10 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_3.h

## See also

[ID3D11UnorderedAccessView](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11UnorderedAccessView1::GetDesc1 method (d3d11\_3.h)

Article 02/22/2024

Gets a description of the resource.

## Syntax

C++

```
void GetDesc1(  
    [out] D3D11_UNORDERED_ACCESS_VIEW_DESC1 *pDesc1  
);
```

## Parameters

[out] pDesc1

Type: [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC1\\*](#)

A pointer to a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC1](#) structure that receives the description of the unordered-access resource.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d11_3.h

Requirement	Value
Library	D3D11.lib

## See also

[ID3D11UnorderedAccessView1](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# ID3D11View interface (d3d11.h)

Article 02/22/2024

A view interface specifies the parts of a resource the pipeline can access during rendering.

## Inheritance

The [ID3D11View](#) interface inherits from [ID3D11DeviceChild](#). [ID3D11View](#) also has these types of members:

## Methods

The [ID3D11View](#) interface has these methods.

[+] [Expand table](#)

### [ID3D11View::GetResource](#)

Get the resource that is accessed through this view. ([ID3D11View::GetResource](#))

## Remarks

A view interface is the base interface for all views. There are four types of views; a depth-stencil view, a render-target view, a shader-resource view, and an unordered-access view.

- To create a render-target view, call [ID3D11Device::CreateRenderTargetView](#).
- To create a depth-stencil view, call [ID3D11Device::CreateDepthStencilView](#).
- To create a shader-resource view, call [ID3D11Device::CreateShaderResourceView](#).
- To create an unordered-access view, call [ID3D11Device::CreateUnorderedAccessView](#).

All resources must be bound to the pipeline before they can be accessed.

- To bind a render-target view or a depth-stencil view, call [ID3D11DeviceContext::OMSetRenderTargets](#).
- To bind a shader resource, call [ID3D11DeviceContext::VSSetShaderResources](#).

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11DeviceChild](#)

[Resource Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11View::GetResource method (d3d11.h)

Article02/22/2024

Get the resource that is accessed through this view.

## Syntax

C++

```
void GetResource(  
    [out] ID3D11Resource **ppResource  
);
```

## Parameters

[out] ppResource

Type: [ID3D11Resource\\*\\*](#)

Address of a pointer to the resource that is accessed through this view. (See [ID3D11Resource](#).)

## Return value

None

## Remarks

This function increments the reference count of the resource by one, so it is necessary to call **Release** on the returned pointer when the application is done with it. Destroying (or losing) the returned pointer before **Release** is called will result in a memory leak.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11View](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Resource Functions (Direct3D 11 Graphics)

Article • 12/10/2020

This section contains information about the resource functions.

## In this section

Topic	Description
<a href="#">D3D11CalcSubresource</a>	Calculates a subresource index for a texture.

## Related topics

[Resource Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11CalcSubresource function (d3d11.h)

Article 02/22/2024

Calculates a subresource index for a texture.

## Syntax

C++

```
UINT D3D11CalcSubresource(  
    UINT MipSlice,  
    UINT ArraySlice,  
    UINT MipLevels  
) ;
```

## Parameters

MipSlice

Type: [UINT](#)

A zero-based index for the mipmap level to address; 0 indicates the first, most detailed mipmap level.

ArraySlice

Type: [UINT](#)

The zero-based index for the array level to address; always use 0 for volume (3D) textures.

MipLevels

Type: [UINT](#)

Number of mipmap levels in the resource.

## Return value

Type: [UINT](#)

The index which equals MipSlice + (ArraySlice \* MipLevels).

## Remarks

A buffer is an unstructured resource and is therefore defined as containing a single subresource. APIs that take buffers do not need a subresource index. A texture on the other hand is highly structured. Each texture object may contain one or more subresources depending on the size of the array and the number of mipmap levels.

For volume (3D) textures, all slices for a given mipmap level are a single subresource index.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
DLL	D3d11.lib

## See also

[Core Functions](#)

[Resource Functions](#)

---

## Feedback

Was this page helpful?

Yes

No

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

# Resource Structures (Direct3D 11 Graphics)

Article • 12/10/2020

Structures are used to create and use resources.

## In this section

Topic	Description
<a href="#">D3D11_BUFFER_DESC</a>	Describes a buffer resource.
<a href="#">D3D11_BUFFER_RTV</a>	Specifies the elements in a buffer resource to use in a render-target view.
<a href="#">D3D11_BUFFER_SRV</a>	Specifies the elements in a buffer resource to use in a shader-resource view.
<a href="#">D3D11_BUFFER_UAV</a>	Describes the elements in a buffer to use in a unordered-access view.
<a href="#">D3D11_BUFFEREX_SRV</a>	Describes the elements in a raw buffer resource to use in a shader-resource view.
<a href="#">D3D11_DEPTH_STENCIL_VIEW_DESC</a>	Specifies the subresources of a texture that are accessible from a depth-stencil view.
<a href="#">D3D11_MAPPED_SUBRESOURCE</a>	Provides access to subresource data.
<a href="#">D3D11_PACKED_MIP_DESC</a>	Describes the tile structure of a tiled resource with mipmaps.
<a href="#">D3D11_RENDER_TARGET_VIEW_DESC</a>	Specifies the subresources from a resource that are accessible using a render-target view.
<a href="#">D3D11_RENDER_TARGET_VIEW_DESC1</a>	Describes the subresources from a resource that are accessible using a render-target view.
<a href="#">D3D11_SHADER_RESOURCE_VIEW_DESC</a>	Describes a shader-resource view.
<a href="#">D3D11_SHADER_RESOURCE_VIEW_DESC1</a>	Describes a shader-resource view.
<a href="#">D3D11_SUBRESOURCE_DATA</a>	Specifies data for initializing a subresource.
<a href="#">D3D11_SUBRESOURCE_TILING</a>	Describes a tiled subresource volume.

Topic	Description
<a href="#">D3D11_TEX1D_ARRAY_DSV</a>	Specifies the subresources from an array of 1D textures to use in a depth-stencil view.
<a href="#">D3D11_TEX1D_ARRAY_RTV</a>	Specifies the subresources from an array of 1D textures to use in a render-target view.
<a href="#">D3D11_TEX1D_ARRAY_SRV</a>	Specifies the subresources from an array of 1D textures to use in a shader-resource view.
<a href="#">D3D11_TEX1D_ARRAY_UAV</a>	Describes an array of unordered-access 1D texture resources.
<a href="#">D3D11_TEX1D_DSV</a>	Specifies the subresource from a 1D texture that is accessible to a depth-stencil view.
<a href="#">D3D11_TEX1D_RTV</a>	Specifies the subresource from a 1D texture to use in a render-target view.
<a href="#">D3D11_TEX1D_SRV</a>	Specifies the subresource from a 1D texture to use in a shader-resource view.
<a href="#">D3D11_TEX1D_UAV</a>	Describes a unordered-access 1D texture resource.
<a href="#">D3D11_TEX2D_ARRAY_DSV</a>	Specifies the subresources from an array 2D textures that are accessible to a depth-stencil view.
<a href="#">D3D11_TEX2D_ARRAY_RTV</a>	Specifies the subresources from an array of 2D textures to use in a render-target view.
<a href="#">D3D11_TEX2D_ARRAY_RTV1</a>	Describes the subresources from an array of 2D textures to use in a render-target view.
<a href="#">D3D11_TEX2D_ARRAY_SRV</a>	Specifies the subresources from an array of 2D textures to use in a shader-resource view.
<a href="#">D3D11_TEX2D_ARRAY_SRV1</a>	Describes the subresources from an array of 2D textures to use in a shader-resource view.
<a href="#">D3D11_TEX2D_ARRAY_UAV</a>	Describes an array of unordered-access 2D texture resources.
<a href="#">D3D11_TEX2D_ARRAY_UAV1</a>	Describes an array of unordered-access 2D texture resources.
<a href="#">D3D11_TEX2D_DSV</a>	Specifies the subresource from a 2D texture that is accessible to a depth-stencil view.
<a href="#">D3D11_TEX2D_RTV</a>	Specifies the subresource from a 2D texture to use in a render-target view.

Topic	Description
<a href="#">D3D11_TEX2D_RTV1</a>	Describes the subresource from a 2D texture to use in a render-target view.
<a href="#">D3D11_TEX2D_SRV</a>	Specifies the subresource from a 2D texture to use in a shader-resource view.
<a href="#">D3D11_TEX2D_SRV1</a>	Describes the subresource from a 2D texture to use in a shader-resource view.
<a href="#">D3D11_TEX2D_UAV</a>	Describes a unordered-access 2D texture resource.
<a href="#">D3D11_TEX2D_UAV1</a>	Describes a unordered-access 2D texture resource.
<a href="#">D3D11_TEX2DMS_ARRAY_DSV</a>	Specifies the subresources from an array of multisampled 2D textures for a depth-stencil view.
<a href="#">D3D11_TEX2DMS_ARRAY_RTV</a>	Specifies the subresources from a an array of multisampled 2D textures to use in a render-target view.
<a href="#">D3D11_TEX2DMS_ARRAY_SRV</a>	Specifies the subresources from an array of multisampled 2D textures to use in a shader-resource view.
<a href="#">D3D11_TEX2DMS_DSV</a>	Specifies the subresource from a multisampled 2D texture that is accessible to a depth-stencil view.
<a href="#">D3D11_TEX2DMS_RTV</a>	Specifies the subresource from a multisampled 2D texture to use in a render-target view.
<a href="#">D3D11_TEX2DMS_SRV</a>	Specifies the subresources from a multisampled 2D texture to use in a shader-resource view.
<a href="#">D3D11_TEX3D_RTV</a>	Specifies the subresources from a 3D texture to use in a render-target view.
<a href="#">D3D11_TEX3D_SRV</a>	Specifies the subresources from a 3D texture to use in a shader-resource view.
<a href="#">D3D11_TEX3D_UAV</a>	Describes a unordered-access 3D texture resource.
<a href="#">D3D11_TEXCUBE_ARRAY_SRV</a>	Specifies the subresources from an array of cube textures to use in a shader-resource view.
<a href="#">D3D11_TEXCUBE_SRV</a>	Specifies the subresource from a cube texture to use in a shader-resource view.
<a href="#">D3D11_TEXTURE1D_DESC</a>	Describes a 1D texture.
<a href="#">D3D11_TEXTURE2D_DESC</a>	Describes a 2D texture.

Topic	Description
D3D11_TEXTURE2D_DESC1	Describes a 2D texture.
D3D11_TEXTURE3D_DESC	Describes a 3D texture.
D3D11_TEXTURE3D_DESC1	Describes a 3D texture.
D3D11_TILE_REGION_SIZE	Describes the size of a tiled region.
D3D11_TILED_RESOURCE_COORDINATE	Describes the coordinates of a tiled resource.
D3D11_TILE_SHAPE	Describes the shape of a tile by specifying its dimensions.
D3D11_UNORDERED_ACCESS_VIEW_DESC	Specifies the subresources from a resource that are accessible using an unordered-access view.
D3D11_UNORDERED_ACCESS_VIEW_DESC1	Describes the subresources from a resource that are accessible using an unordered-access view.

## Related topics

[Resource Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BUFFER\_DESC structure (d3d11.h)

Article 07/27/2022

Describes a buffer resource.

## Syntax

C++

```
typedef struct D3D11_BUFFER_DESC {
    UINT          ByteWidth;
    D3D11_USAGE   Usage;
    UINT          BindFlags;
    UINT          CPUAccessFlags;
    UINT          MiscFlags;
    UINT          StructureByteStride;
} D3D11_BUFFER_DESC;
```

## Members

`ByteWidth`

Type: [UINT](#)

Size of the buffer in bytes.

`Usage`

Type: [D3D11\\_USAGE](#)

Identify how the buffer is expected to be read from and written to. Frequency of update is a key factor. The most common value is typically `D3D11_USAGE_DEFAULT`; see [D3D11\\_USAGE](#) for all possible values.

`BindFlags`

Type: [UINT](#)

Identify how the buffer will be bound to the pipeline. Flags (see [D3D11\\_BIND\\_FLAG](#)) can be combined with a bitwise OR.

`CPUAccessFlags`

Type: [UINT](#)

CPU access flags (see [D3D11\\_CPU\\_ACCESS\\_FLAG](#)) or 0 if no CPU access is necessary. Flags can be combined with a bitwise OR.

#### MiscFlags

Type: [UINT](#)

Miscellaneous flags (see [D3D11\\_RESOURCE\\_MISC\\_FLAG](#)) or 0 if unused. Flags can be combined with a bitwise OR.

#### StructureByteStride

Type: [UINT](#)

The size of each element in the buffer structure (in bytes) when the buffer represents a structured buffer. For more info about structured buffers, see [Structured Buffer](#).

The size value in **StructureByteStride** must match the size of the format that you use for views of the buffer. For example, if you use a shader resource view (SRV) to read a buffer in a pixel shader, the SRV format size must match the size value in **StructureByteStride**.

## Remarks

This structure is used by [ID3D11Device::CreateBuffer](#) to create buffer resources.

In addition to this structure, you can also use the [CD3D11\\_BUFFER\\_DESC](#) derived structure, which is defined in D3D11.h and behaves like an inherited class, to help create a buffer description.

If the bind flag is [D3D11\\_BIND\\_CONSTANT\\_BUFFER](#), you must set the **ByteWidth** value in multiples of 16, and less than or equal to

[D3D11\\_REQ\\_CONSTANT\\_BUFFER\\_ELEMENT\\_COUNT](#).

## Requirements

Header	d3d11.h
--------	---------

## See also

[Resource Structures](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BUFFER\_RTV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_BUFFER_RTV {
    union {
        UINT FirstElement;
        UINT ElementOffset;
    };
    union {
        UINT NumElements;
        UINT ElementWidth;
    };
} D3D11_BUFFER_RTV;
```

## Members

`FirstElement`

Type: [UINT](#)

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

`ElementOffset`

Type: [UINT](#)

The offset of the first element in the view to access, relative to element 0.

`NumElements`

Type: [UINT](#)

The total number of elements in the view.

`ElementWidth`

Type: [UINT](#)

The width of each element (in bytes). This can be determined from the format stored in the render-target-view description.

## Remarks

A render-target view is a member of a render-target-view description (see [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)). Create a render-target view by calling [ID3D11Device::CreateRenderTargetView](#).

## Requirements

  [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BUFFER\_SRV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_BUFFER_SRV {
    union {
        UINT FirstElement;
        UINT ElementOffset;
    };
    union {
        UINT NumElements;
        UINT ElementWidth;
    };
} D3D11_BUFFER_SRV;
```

## Members

`FirstElement`

Type: **UINT**

Index of the first element to access.

`ElementOffset`

Type: **UINT**

The offset of the first element in the view to access, relative to element 0.

`NumElements`

Type: **UINT**

The total number of elements in the view.

`ElementWidth`

Type: **UINT**

The width of each element (in bytes). This can be determined from the format stored in the shader-resource-view description.

## Remarks

The `D3D11_BUFFER_SRV` structure is a member of the `D3D11_SHADER_RESOURCE_VIEW_DESC` structure, which represents a shader-resource view description. You can create a shader-resource view by calling the `ID3D11Device::CreateShaderResourceView` method.

## Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_BUFFER\_UAV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_BUFFER_UAV {
    UINT FirstElement;
    UINT NumElements;
    UINT Flags;
} D3D11_BUFFER_UAV;
```

## Members

`FirstElement`

Type: [UINT](#)

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

`NumElements`

Type: [UINT](#)

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

`Flags`

Type: [UINT](#)

View options for the resource (see [D3D11\\_BUFFER\\_UAV\\_FLAG](#)).

## Remarks

This structure is used by a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BUFFEREX\_SRV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_BUFFEREX_SRV {
    UINT FirstElement;
    UINT NumElements;
    UINT Flags;
} D3D11_BUFFEREX_SRV;
```

## Members

FirstElement

Type: [UINT](#)

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

NumElements

Type: [UINT](#)

The number of elements in the resource.

Flags

Type: [UINT](#)

A [D3D11\\_BUFFEREX\\_SRV\\_FLAG](#)-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 [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) to create a raw view of a buffer.

## Requirements

[Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_DEPTH\_STENCIL\_VIEW\_DESC structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_DEPTH_STENCIL_VIEW_DESC {
    DXGI_FORMAT          Format;
    D3D11_DSV_DIMENSION ViewDimension;
    UINT                 Flags;
    union {
        D3D11_TEX1D_DSV      Texture1D;
        D3D11_TEX1D_ARRAY_DSV Texture1DArray;
        D3D11_TEX2D_DSV      Texture2D;
        D3D11_TEX2D_ARRAY_DSV Texture2DArray;
        D3D11_TEX2DMS_DSV    Texture2DMS;
        D3D11_TEX2DMS_ARRAY_DSV Texture2DMSArray;
    };
} D3D11_DEPTH_STENCIL_VIEW_DESC;
```

## Members

Format

Type: [DXGI\\_FORMAT](#)

Resource data format (see [DXGI\\_FORMAT](#)). See remarks for allowable formats.

ViewDimension

Type: [D3D11\\_DSV\\_DIMENSION](#)

Type of resource (see [D3D11\\_DSV\\_DIMENSION](#)). Specifies how a depth-stencil resource will be accessed; the value is stored in the union in this structure.

Flags

Type: [UINT](#)

A value that describes whether the texture is read only. Pass 0 to specify that it is not read only; otherwise, pass one of the members of the [D3D11\\_DSV\\_FLAG](#) enumerated type.

#### Texture1D

Type: [D3D11\\_TEX1D\\_DSV](#)

Specifies a 1D texture subresource (see [D3D11\\_TEX1D\\_DSV](#)).

#### Texture1DArray

Type: [D3D11\\_TEX1D\\_ARRAY\\_DSV](#)

Specifies an array of 1D texture subresources (see [D3D11\\_TEX1D\\_ARRAY\\_DSV](#)).

#### Texture2D

Type: [D3D11\\_TEX2D\\_DSV](#)

Specifies a 2D texture subresource (see [D3D11\\_TEX2D\\_DSV](#)).

#### Texture2DArray

Type: [D3D11\\_TEX2D\\_ARRAY\\_DSV](#)

Specifies an array of 2D texture subresources (see [D3D11\\_TEX2D\\_ARRAY\\_DSV](#)).

#### Texture2DMS

Type: [D3D11\\_TEX2DMS\\_DSV](#)

Specifies a multisampled 2D texture (see [D3D11\\_TEX2DMS\\_DSV](#)).

#### Texture2DMSArray

Type: [D3D11\\_TEX2DMS\\_ARRAY\\_DSV](#)

Specifies an array of multisampled 2D textures (see [D3D11\\_TEX2DMS\\_ARRAY\\_DSV](#)).

## 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 cannot use a typeless format. If the format chosen is DXGI\_FORMAT\_UNKNOWN, then the format of the parent resource is used.

A depth-stencil-view description is needed when calling [ID3D11Device::CreateDepthStencilView](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_MAPPED\_SUBRESOURCE structure (d3d11.h)

Article 04/02/2021

Provides access to subresource data.

## Syntax

C++

```
typedef struct D3D11_MAPPED_SUBRESOURCE {
    void *pData;
    UINT RowPitch;
    UINT DepthPitch;
} D3D11_MAPPED_SUBRESOURCE;
```

## Members

pData

Type: **void\***

Pointer to the data. When [ID3D11DeviceContext::Map](#) provides the pointer, the runtime ensures that the pointer has a specific alignment, depending on the following feature levels:

- For [D3D\\_FEATURE\\_LEVEL\\_10\\_0](#) and higher, the pointer is aligned to 16 bytes.
- For lower than [D3D\\_FEATURE\\_LEVEL\\_10\\_0](#), the pointer is aligned to 4 bytes.

RowPitch

Type: **UINT**

The row pitch, or width, or physical size (in bytes) of the data.

DepthPitch

Type: **UINT**

The depth pitch, or width, or physical size (in bytes) of the data.

# Remarks

This structure is used in a call to [ID3D11DeviceContext::Map](#).

The values in these members tell you how much data you can view:

- **pData** points to row 0 and depth slice 0.
- **RowPitch** contains the value that the runtime adds to **pData** to move from row to row, where each row contains multiple pixels.
- **DepthPitch** contains the value that the runtime adds to **pData** to move from depth slice to depth slice, where each depth slice contains multiple rows.

When **RowPitch** and **DepthPitch** are not appropriate for the resource type, the runtime might set their values to 0. So, don't use these values for anything other than iterating over rows and depth. Here are some examples:

- For [Buffer](#) and [Texture1D](#), the runtime assigns values that aren't 0 to **RowPitch** and **DepthPitch**. For example, if a [Buffer](#) contains 8 bytes, the runtime assigns values to **RowPitch** and **DepthPitch** that are greater than or equal to 8.
- For [Texture2D](#), the runtime still assigns a value that isn't 0 to **DepthPitch**, assuming that the field isn't used.

**Note** The runtime might assign values to **RowPitch** and **DepthPitch** that are larger than anticipated because there might be padding between rows and depth.

# Requirements

Header	d3d11.h

## See also

[Resource Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3D11\_PACKED\_MIP\_DESC structure (d3d11\_2.h)

Article07/27/2022

Describes the tile structure of a tiled resource with mipmaps.

## Syntax

C++

```
typedef struct D3D11_PACKED_MIP_DESC {
    UINT8 NumStandardMips;
    UINT8 NumPackedMips;
    UINT  NumTilesForPackedMips;
    UINT  StartTileIndexInOverallResource;
} D3D11_PACKED_MIP_DESC;
```

## Members

**NumStandardMips**

Number of standard mipmaps in the tiled resource.

**NumPackedMips**

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**

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**

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.

## Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Header	d3d11_2.h

## See also

[Resource Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_RENDER\_TARGET\_VIEW\_DESC structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_RENDER_TARGET_VIEW_DESC {
    DXGI_FORMAT           Format;
    D3D11_RTV_DIMENSION ViewDimension;
    union {
        D3D11_BUFFER_RTV      Buffer;
        D3D11_TEX1D_RTV       Texture1D;
        D3D11_TEX1D_ARRAY_RTV Texture1DArray;
        D3D11_TEX2D_RTV       Texture2D;
        D3D11_TEX2D_ARRAY_RTV Texture2DArray;
        D3D11_TEX2DMS_RTV     Texture2DMS;
        D3D11_TEX2DMS_ARRAY_RTV Texture2DMSArray;
        D3D11_TEX3D_RTV       Texture3D;
    };
} D3D11_RENDER_TARGET_VIEW_DESC;
```

## Members

Format

Type: [DXGI\\_FORMAT](#)

The data format (see [DXGI\\_FORMAT](#)).

ViewDimension

Type: [D3D11\\_RTV\\_DIMENSION](#)

The resource type (see [D3D11\\_RTV\\_DIMENSION](#)), which specifies how the render-target resource will be accessed.

Buffer

Type: [D3D11\\_BUFFER\\_RTV](#)

Specifies which buffer elements can be accessed (see [D3D11\\_BUFFER\\_RTV](#)).

Texture1D

Type: [D3D11\\_TEX1D\\_RTV](#)

Specifies the subresources in a 1D texture that can be accessed (see [D3D11\\_TEX1D\\_RTV](#)).

Texture1DArray

Type: [D3D11\\_TEX1D\\_ARRAY\\_RTV](#)

Specifies the subresources in a 1D texture array that can be accessed (see [D3D11\\_TEX1D\\_ARRAY\\_RTV](#)).

Texture2D

Type: [D3D11\\_TEX2D\\_RTV](#)

Specifies the subresources in a 2D texture that can be accessed (see [D3D11\\_TEX2D\\_RTV](#)).

Texture2DArray

Type: [D3D11\\_TEX2D\\_ARRAY\\_RTV](#)

Specifies the subresources in a 2D texture array that can be accessed (see [D3D11\\_TEX2D\\_ARRAY\\_RTV](#)).

Texture2DMS

Type: [D3D11\\_TEX2DMS\\_RTV](#)

Specifies a single subresource because a multisampled 2D texture only contains one subresource (see [D3D11\\_TEX2DMS\\_RTV](#)).

Texture2DMSArray

Type: [D3D11\\_TEX2DMS\\_ARRAY\\_RTV](#)

Specifies the subresources in a multisampled 2D texture array that can be accessed (see [D3D11\\_TEX2DMS\\_ARRAY\\_RTV](#)).

Texture3D

Type: [D3D11\\_TEX3D\\_RTV](#)

Specifies subresources in a 3D texture that can be accessed (see [D3D11\\_TEX3D\\_RTV](#)).

## Remarks

A render-target-view description is passed into [ID3D11Device::CreateRenderTargetView](#) to create a render target.

A render-target-view cannot use the following formats:

- Any typeless format.
- DXGI\_FORMAT\_R32G32B32 if the view will be used to bind a buffer (vertex, index, constant, or stream-output).

If the format is set to DXGI\_FORMAT\_UNKNOWN, then the format of the resource that the view binds to the pipeline will be used.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# CD3D11\_RENDER\_TARGET\_VIEW\_DESC1 structure (d3d11\_3.h)

Article02/16/2023

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

## Syntax

C++

```
struct CD3D11_RENDER_TARGET_VIEW_DESC1 : D3D11_RENDER_TARGET_VIEW_DESC1 {
    void CD3D11_RENDER_TARGET_VIEW_DESC1();
    void CD3D11_RENDER_TARGET_VIEW_DESC1(
        const D3D11_RENDER_TARGET_VIEW_DESC1 & o
    );
    void CD3D11_RENDER_TARGET_VIEW_DESC1(
        D3D11_RTV_DIMENSION viewDimension,
        DXGI_FORMAT format,
        UINT mipSlice,
        UINT firstArraySlice,
        UINT arraySize,
        UINT planeSlice
    );
    void CD3D11_RENDER_TARGET_VIEW_DESC1(
        ID3D11Buffer *unnamedParam1,
        DXGI_FORMAT format,
        UINT firstElement,
        UINT numElements
    );
    void CD3D11_RENDER_TARGET_VIEW_DESC1(
        ID3D11Texture1D *pTex1D,
        D3D11_RTV_DIMENSION viewDimension,
        DXGI_FORMAT format,
        UINT mipSlice,
        UINT firstArraySlice,
        UINT arraySize
    );
    void CD3D11_RENDER_TARGET_VIEW_DESC1(
        ID3D11Texture2D *pTex2D,
        D3D11_RTV_DIMENSION viewDimension,
        DXGI_FORMAT format,
        UINT mipSlice,
        UINT firstArraySlice,
        UINT arraySize,
        UINT planeSlice
    );
    void CD3D11_RENDER_TARGET_VIEW_DESC1(
```

```
    ID3D11Texture3D *pTex3D,
    DXGI_FORMAT      format,
    UINT             mipSlice,
    UINT             firstWSlice,
    UINT             wSize
);
void ~CD3D11_RENDER_TARGET_VIEW_DESC1();
};
```

## Inheritance

The **CD3D11\_RENDER\_TARGET\_VIEW\_DESC1** structure implements **D3D11\_RENDER\_TARGET\_VIEW\_DESC1**.

## Members

```
void CD3D11_RENDER_TARGET_VIEW_DESC1()
```

TBD

```
void CD3D11_RENDER_TARGET_VIEW_DESC1( const D3D11_RENDER_TARGET_VIEW_DESC1 & o)
```

```
void CD3D11_RENDER_TARGET_VIEW_DESC1( D3D11_RTV_DIMENSION viewDimension,
DXGI_FORMAT format, UINT mipSlice, UINT firstArraySlice, UINT arraySize, UINT
planeSlice)
```

```
void CD3D11_RENDER_TARGET_VIEW_DESC1( ID3D11Buffer *unnamedParam1, DXGI_FORMAT
format, UINT firstElement, UINT numElements)
```

```
void CD3D11_RENDER_TARGET_VIEW_DESC1( ID3D11Texture1D *pTex1D, D3D11_RTV_DIMENSION
viewDimension, DXGI_FORMAT format, UINT mipSlice, UINT firstArraySlice, UINT
arraySize)
```

```
void CD3D11_RENDER_TARGET_VIEW_DESC1( ID3D11Texture2D *pTex2D, D3D11_RTV_DIMENSION
viewDimension, DXGI_FORMAT format, UINT mipSlice, UINT firstArraySlice, UINT
arraySize, UINT planeSlice)
```

```
void CD3D11_RENDER_TARGET_VIEW_DESC1( ID3D11Texture3D *pTex3D, DXGI_FORMAT format,
UINT mipSlice, UINT firstWSlice, UINT wSize)
```

```
void ~CD3D11_RENDER_TARGET_VIEW_DESC1()
```

TBD

## Remarks

A render-target-view description is passed into [ID3D11Device3::CreateRenderTargetView1](#) to create a render target.

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	d3d11_3.h
--------	-----------

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SHADER\_RESOURCE\_VIEW\_DESC structure (d3d11.h)

Article 07/27/2022

Describes a shader-resource view.

## Syntax

C++

```
typedef struct D3D11_SHADER_RESOURCE_VIEW_DESC {
    DXGI_FORMAT Format;
    D3D11_SRV_DIMENSION ViewDimension;
    union {
        D3D11_BUFFER_SRV Buffer;
        D3D11_TEX1D_SRV Texture1D;
        D3D11_TEX1D_ARRAY_SRV Texture1DArray;
        D3D11_TEX2D_SRV Texture2D;
        D3D11_TEX2D_ARRAY_SRV Texture2DArray;
        D3D11_TEX2DMS_SRV Texture2DMS;
        D3D11_TEX2DMS_ARRAY_SRV Texture2DMSArray;
        D3D11_TEX3D_SRV Texture3D;
        D3D11_TEXCUBE_SRV TextureCube;
        D3D11_TEXCUBE_ARRAY_SRV TextureCubeArray;
        D3D11_BUFFEREX_SRV BufferEx;
    };
} D3D11_SHADER_RESOURCE_VIEW_DESC;
```

## Members

Format

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#) specifying the viewing format. See remarks.

ViewDimension

Type: [D3D11\\_SRV\\_DIMENSION](#)

The resource type of the view. See [D3D11\\_SRV\\_DIMENSION](#). You must set *ViewDimension* to the same resource type as that of the underlying resource. This parameter also determines which \_SRV to use in the union below.

## Buffer

Type: [D3D11\\_BUFFER\\_SRV](#)

View the resource as a buffer using information from a shader-resource view (see [D3D11\\_BUFFER\\_SRV](#)).

## Texture1D

Type: [D3D11\\_TEX1D\\_SRV](#)

View the resource as a 1D texture using information from a shader-resource view (see [D3D11\\_TEX1D\\_SRV](#)).

## Texture1DArray

Type: [D3D11\\_TEX1D\\_ARRAY\\_SRV](#)

View the resource as a 1D-texture array using information from a shader-resource view (see [D3D11\\_TEX1D\\_ARRAY\\_SRV](#)).

## Texture2D

Type: [D3D11\\_TEX2D\\_SRV](#)

View the resource as a 2D-texture using information from a shader-resource view (see [D3D11\\_TEX2D\\_SRV](#)).

## Texture2DArray

Type: [D3D11\\_TEX2D\\_ARRAY\\_SRV](#)

View the resource as a 2D-texture array using information from a shader-resource view (see [D3D11\\_TEX2D\\_ARRAY\\_SRV](#)).

## Texture2DMS

Type: [D3D11\\_TEX2DMS\\_SRV](#)

View the resource as a 2D-multisampled texture using information from a shader-resource view (see [D3D11\\_TEX2DMS\\_SRV](#)).

## Texture2DMSArray

Type: [D3D11\\_TEX2DMS\\_ARRAY\\_SRV](#)

View the resource as a 2D-multisampled-texture array using information from a shader-resource view (see [D3D11\\_TEX2DMS\\_ARRAY\\_SRV](#)).

#### Texture3D

Type: [D3D11\\_TEX3D\\_SRV](#)

View the resource as a 3D texture using information from a shader-resource view (see [D3D11\\_TEX3D\\_SRV](#)).

#### TextureCube

Type: [D3D11\\_TEXCUBE\\_SRV](#)

View the resource as a 3D-cube texture using information from a shader-resource view (see [D3D11\\_TEXCUBE\\_SRV](#)).

#### TextureCubeArray

Type: [D3D11\\_TEXCUBE\\_ARRAY\\_SRV](#)

View the resource as a 3D-cube-texture array using information from a shader-resource view (see [D3D11\\_TEXCUBE\\_ARRAY\\_SRV](#)).

#### BufferEx

Type: [D3D11\\_BUFFEREX\\_SRV](#)

View the resource as a raw buffer using information from a shader-resource view (see [D3D11\\_BUFFEREX\\_SRV](#)). For more info about raw viewing of buffers, see [Raw Views of Buffers](#).

## 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 cannot 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 [ID3D11Device::CreateShaderResourceView](#). To view a shader-resource-view description, call [ID3D11ShaderResourceView::GetDesc](#).

## Requirements

Header	d3d11.h (include D3D11Shader.h)
--------	---------------------------------

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC C1 structure (d3d11\_3.h)

Article02/16/2023

Describes a shader-resource view.

## Syntax

C++

```
struct CD3D11_SHADER_RESOURCE_VIEW_DESC1 : D3D11_SHADER_RESOURCE_VIEW_DESC1
{
    void CD3D11_SHADER_RESOURCE_VIEW_DESC1();
    void CD3D11_SHADER_RESOURCE_VIEW_DESC1(
        const D3D11_SHADER_RESOURCE_VIEW_DESC1 & o
    );
    void CD3D11_SHADER_RESOURCE_VIEW_DESC1(
        D3D11_SRV_DIMENSION viewDimension,
        DXGI_FORMAT format,
        UINT mostDetailedMip,
        UINT mipLevels,
        UINT firstArraySlice,
        UINT arraySize,
        UINT flags,
        UINT planeSlice
    );
    void CD3D11_SHADER_RESOURCE_VIEW_DESC1(
        ID3D11Buffer *unnamedParam1,
        DXGI_FORMAT format,
        UINT firstElement,
        UINT numElements,
        UINT flags
    );
    void CD3D11_SHADER_RESOURCE_VIEW_DESC1(
        ID3D11Texture1D *pTex1D,
        D3D11_SRV_DIMENSION viewDimension,
        DXGI_FORMAT format,
        UINT mostDetailedMip,
        UINT mipLevels,
        UINT firstArraySlice,
        UINT arraySize
    );
    void CD3D11_SHADER_RESOURCE_VIEW_DESC1(
        ID3D11Texture2D *pTex2D,
        D3D11_SRV_DIMENSION viewDimension,
        DXGI_FORMAT format,
        UINT mostDetailedMip,
        UINT mipLevels,
        UINT firstArraySlice,
```

```

        UINT           arraySize,
        UINT           planeSlice
    );
void CD3D11_SHADER_RESOURCE_VIEW_DESC1(
    ID3D11Texture3D *pTex3D,
    DXGI_FORMAT      format,
    UINT             mostDetailedMip,
    UINT             mipLevels
);
void ~CD3D11_SHADER_RESOURCE_VIEW_DESC1();
};

```

## Inheritance

The `CD3D11_SHADER_RESOURCE_VIEW_DESC1` structure implements `D3D11_SHADER_RESOURCE_VIEW_DESC1`.

## Members

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC1()
```

TBD

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC1( const D3D11_SHADER_RESOURCE_VIEW_DESC1 &
o)
```

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC1( D3D11_SRV_DIMENSION viewDimension,
DXGI_FORMAT format, UINT mostDetailedMip, UINT mipLevels, UINT firstArraySlice,
UINT arraySize, UINT flags, UINT planeSlice)
```

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC1( ID3D11Buffer *unnamedParam1, DXGI_FORMAT
format, UINT firstElement, UINT numElements, UINT flags)
```

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC1( ID3D11Texture1D *pTex1D,
D3D11_SRV_DIMENSION viewDimension, DXGI_FORMAT format, UINT mostDetailedMip, UINT
mipLevels, UINT firstArraySlice, UINT arraySize)
```

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC1( ID3D11Texture2D *pTex2D,
D3D11_SRV_DIMENSION viewDimension, DXGI_FORMAT format, UINT mostDetailedMip, UINT
mipLevels, UINT firstArraySlice, UINT arraySize, UINT planeSlice)
```

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC1( ID3D11Texture3D *pTex3D, DXGI_FORMAT
format, UINT mostDetailedMip, UINT mipLevels)
```

```
void ~CD3D11_SHADER_RESOURCE_VIEW_DESC1()
```

TBD

## 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 cannot 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 [ID3D11Device3::CreateShaderResourceView1](#). To view a shader-resource-view description, call [ID3D11ShaderResourceView1::GetDesc1](#).

## Requirements

Header	d3d11_3.h
--------	-----------

## See also

[Resource Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SUBRESOURCE\_DATA structure (d3d11.h)

Article 02/22/2024

Specifies data for initializing a subresource.

## Syntax

C++

```
typedef struct D3D11_SUBRESOURCE_DATA {
    const void *pSysMem;
    UINT         SysMemPitch;
    UINT         SysMemSlicePitch;
} D3D11_SUBRESOURCE_DATA;
```

## Members

pSysMem

Type: **const void\***

Pointer to the initialization data.

SysMemPitch

Type: **UINT**

The distance (in bytes) from the beginning of one line of a texture to the next line. System-memory pitch is used only for 2D and 3D texture data as it has no meaning for the other resource types. Specify the distance from the first pixel of one 2D slice of a 3D texture to the first pixel of the next 2D slice in that texture in the **SysMemSlicePitch** member.

SysMemSlicePitch

Type: **UINT**

The distance (in bytes) from the beginning of one depth level to the next. System-memory-slice pitch is only used for 3D texture data as it has no meaning for the other resource types.

## Remarks

This structure is used in calls to create buffers ([ID3D11Device::CreateBuffer](#)) and textures ([ID3D11Device::CreateTexture1D](#), [ID3D11Device::CreateTexture2D](#), and

[ID3D11Device::CreateTexture3D](#)). If the resource you create does not require a system-memory pitch or a system-memory-slice pitch, you can use those members to pass size information, which might help you when you debug a problem with creating a resource.

A subresource is a single mipmap-level surface. You can pass an array of subresources to one of the preceding methods to create the resource. A subresource can be 1D, 2D, or 3D. How you set the members of **D3D11\_SUBRESOURCE\_DATA** depend on whether the subresource is 1D, 2D, or 3D.

The x, y, and d values are 0-based indices and **BytesPerPixel** depends on the pixel format. For mipmapped 3D surfaces, the number of depth slices in each level is half the number of the previous level (minimum 1) and rounded down if dividing by two results in a non-whole number.

**Note** An application must not rely on **SysMemPitch** being exactly equal to the number of texels in a line times the size of a texel. In some cases, **SysMemPitch** will include padding to skip past additional data in a line. This could be padding for alignment or the texture could be a subsection of a larger texture. For example, the **D3D11\_SUBRESOURCE\_DATA** structure could represent a 32 by 32 subsection of a 128 by 128 texture. The value for **SysMemSlicePitch** will reflect any padding included in **SysMemPitch**.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SUBRESOURCE\_TILING structure (d3d11\_2.h)

Article 02/22/2024

Describes a tiled subresource volume.

## Syntax

C++

```
typedef struct D3D11_SUBRESOURCE_TILING {
    UINT    WidthInTiles;
    UINT16  HeightInTiles;
    UINT16  DepthInTiles;
    UINT    StartTileIndexInOverallResource;
} D3D11_SUBRESOURCE_TILING;
```

## Members

`WidthInTiles`

Type: [UINT](#)

The width in tiles of the subresource.

`HeightInTiles`

Type: [UINT16](#)

The height in tiles of the subresource.

`DepthInTiles`

Type: [UINT16](#)

The depth in tiles of the subresource.

`StartTileIndexInOverallResource`

Type: [UINT](#)

The index of the tile in the overall tiled subresource to start with.

[GetResourceTiling](#) sets `StartTileIndexInOverallResource` to `D3D11_PACKED_TILE` (0xffffffff) to indicate that the whole `D3D11_SUBRESOURCE_TILING` structure is meaningless, and the info to which the `pPackedMipDesc` parameter of [GetResourceTiling](#) points applies. For packed tiles, the description of the packed mipmaps comes from a `D3D11_PACKED_MIP_DESC` structure instead.

## Remarks

Each packed mipmap is individually reported as 0 for `WidthInTiles`, `HeightInTiles` and `DepthInTiles`.

The total number of tiles in subresources is `WidthInTiles * HeightInTiles * DepthInTiles`.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11_2.h

## See also

[Resource Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX1D\_ARRAY\_DSV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX1D_ARRAY_DSV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX1D_ARRAY_DSV;
```

## Members

MipSlice

Type: [UINT](#)

The index of the first mipmap level to use.

FirstArraySlice

Type: [UINT](#)

The index of the first texture to use in an array of textures.

ArraySize

Type: [UINT](#)

Number of textures to use.

## Remarks

This structure is one member of a depth-stencil-view description (see [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)).

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX1D\_ARRAY\_RTV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX1D_ARRAY_RTV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX1D_ARRAY_RTV;
```

## Members

MipSlice

Type: [UINT](#)

The index of the mipmap level to use mip slice.

FirstArraySlice

Type: [UINT](#)

The index of the first texture to use in an array of textures.

ArraySize

Type: [UINT](#)

Number of textures to use.

## Remarks

This structure is one member of a render-target-view description (see [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)).

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX1D\_ARRAY\_SRV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX1D_ARRAY_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX1D_ARRAY_SRV;
```

## Members

`MostDetailedMip`

Type: [UINT](#)

Index of the most detailed mipmap level to use; this number is between 0 and [MipLevels](#) (from the original Texture1D for which [ID3D11Device::CreateShaderResourceView](#) creates a view) -1.

`MipLevels`

Type: [UINT](#)

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#).

Set to -1 to indicate all the mipmap levels from [MostDetailedMip](#) on down to least detailed.

`FirstArraySlice`

Type: [UINT](#)

The index of the first texture to use in an array of textures.

### ArraySize

Type: [UINT](#)

Number of textures in the array.

## Remarks

This structure is one member of a shader-resource-view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)).

## Requirements

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

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX1D\_ARRAY\_UAV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX1D_ARRAY_UAV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX1D_ARRAY_UAV;
```

## Members

MipSlice

Type: [UINT](#)

The mipmap slice index.

FirstArraySlice

Type: [UINT](#)

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

ArraySize

Type: [UINT](#)

The number of slices in the array.

## Remarks

This structure is used by a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX1D\_DSV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX1D_DSV {  
    UINT MipSlice;  
} D3D11_TEX1D_DSV;
```

## Members

MipSlice

Type: [UINT](#)

The index of the first mipmap level to use.

## Remarks

This structure is one member of a depth-stencil-view description (see [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX1D\_RTV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX1D_RTV {
    UINT MipSlice;
} D3D11_TEX1D_RTV;
```

## Members

MipSlice

Type: [UINT](#)

The index of the mipmap level to use mip slice.

## Remarks

This structure is one member of a render-target-view description (see [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX1D\_SRV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX1D_SRV {  
    UINT MostDetailedMip;  
    UINT MipLevels;  
} D3D11_TEX1D_SRV;
```

## Members

**MostDetailedMip**

Type: [UINT](#)

Index of the most detailed mipmap level to use; this number is between 0 and **MipLevels** (from the original Texture1D for which [ID3D11Device::CreateShaderResourceView](#) creates a view) -1.

**MipLevels**

Type: [UINT](#)

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.

## Remarks

This structure is one member of a shader-resource-view description (see [D3D11\\_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 [ID3D11Device::CreateShaderResourceView](#) creates the view. In this situation,

**MostDetailedMip** is greater than the **MipLevels** in the view. However, **MostDetailedMip** is not greater than the **MipLevels** in the original resource.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX1D\_UAV structure (d3d11.h)

Article 02/22/2024

Describes a unordered-access 1D texture resource.

## Syntax

C++

```
typedef struct D3D11_TEX1D_UAV {
    UINT MipSlice;
} D3D11_TEX1D_UAV;
```

## Members

MipSlice

Type: [UINT](#)

The mipmap slice index.

## Remarks

This structure is used by a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_ARRAY\_DSV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_ARRAY_DSV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX2D_ARRAY_DSV;
```

## Members

`MipSlice`

Type: [UINT](#)

The index of the first mipmap level to use.

`FirstArraySlice`

Type: [UINT](#)

The index of the first texture to use in an array of textures.

`ArraySize`

Type: [UINT](#)

Number of textures to use.

## Remarks

This structure is one member of a depth-stencil-view description (see [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)).

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_ARRAY\_RTV structure (d3d11.h)

Article04/02/2021

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_ARRAY_RTV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX2D_ARRAY_RTV;
```

## Members

MipSlice

Type: [UINT](#)

The index of the mipmap level to use mip slice.

FirstArraySlice

Type: [UINT](#)

The index of the first texture to use in an array of textures.

ArraySize

Type: [UINT](#)

Number of textures in the array to use in the render target view, starting from [FirstArraySlice](#).

## Remarks

This structure is one member of a render-target-view description (see [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)).

# Requirements

Header	d3d11.h
--------	---------

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_ARRAY\_RTV1 structure (d3d11\_3.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_ARRAY_RTV1 {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
    UINT PlaneSlice;
} D3D11_TEX2D_ARRAY_RTV1;
```

## 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.

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_TEX2D\_ARRAY\_SRV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_ARRAY_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX2D_ARRAY_SRV;
```

## Members

`MostDetailedMip`

Type: [UINT](#)

Index of the most detailed mipmap level to use; this number is between 0 and [MipLevels](#) (from the original Texture2D for which [ID3D11Device::CreateShaderResourceView](#) creates a view) -1.

`MipLevels`

Type: [UINT](#)

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#).

Set to -1 to indicate all the mipmap levels from [MostDetailedMip](#) on down to least detailed.

`FirstArraySlice`

Type: [UINT](#)

The index of the first texture to use in an array of textures.

### ArraySize

Type: [UINT](#)

Number of textures in the array.

## Remarks

This structure is one member of a shader-resource-view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)).

## Requirements

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

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_ARRAY\_SRV1 structure (d3d11\_3.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_ARRAY_SRV1 {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT FirstArraySlice;
    UINT ArraySize;
    UINT PlaneSlice;
} D3D11_TEX2D_ARRAY_SRV1;
```

## Members

**MostDetailedMip**

Index of the most detailed mipmap level to use; this number is between 0 and (**MipLevels** (from the original Texture2D for which [ID3D11Device3::CreateShaderResourceView1](#) creates a view) - 1).

**MipLevels**

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D11\\_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.

## Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_ARRAY\_UAV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_ARRAY_UAV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX2D_ARRAY_UAV;
```

## Members

MipSlice

Type: [UINT](#)

The mipmap slice index.

FirstArraySlice

Type: [UINT](#)

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

ArraySize

Type: [UINT](#)

The number of slices in the array.

## Remarks

This structure is used by a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_ARRAY\_UAV1 structure (d3d11\_3.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_ARRAY_UAV1 {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
    UINT PlaneSlice;
} D3D11_TEX2D_ARRAY_UAV1;
```

## 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.

## Requirements

[] Expand table

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_TEX2D\_DSV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_DSV {  
    UINT MipSlice;  
} D3D11_TEX2D_DSV;
```

## Members

MipSlice

Type: [UINT](#)

The index of the first mipmap level to use.

## Remarks

This structure is one member of a depth-stencil-view description (see [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_RTV structure (d3d11.h)

Article 02/22/2024

Specifies the subresource from a 2D texture to use in a render-target view.

## Syntax

C++

```
typedef struct D3D11_TEX2D_RTV {
    UINT MipSlice;
} D3D11_TEX2D_RTV;
```

## Members

MipSlice

Type: [UINT](#)

The index of the mipmap level to use mip slice.

## Remarks

This structure is one member of a render-target-view description (see [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_RTV1 structure (d3d11\_3.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_RTV1 {
    UINT MipSlice;
    UINT PlaneSlice;
} D3D11_TEX2D_RTV1;
```

## Members

MipSlice

The index of the mipmap level to use mip slice.

PlaneSlice

The index (plane slice number) of the plane to use in the texture.

## Requirements

 Expand table

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_SRV structure (d3d11.h)

Article 02/22/2024

Specifies the subresource from a 2D texture to use in a shader-resource view.

## Syntax

C++

```
typedef struct D3D11_TEX2D_SRV {  
    UINT MostDetailedMip;  
    UINT MipLevels;  
} D3D11_TEX2D_SRV;
```

## Members

`MostDetailedMip`

Type: [UINT](#)

Index of the most detailed mipmap level to use; this number is between 0 and [MipLevels](#) (from the original Texture2D for which [ID3D11Device::CreateShaderResourceView](#) creates a view) -1.

`MipLevels`

Type: [UINT](#)

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#).

Set to -1 to indicate all the mipmap levels from [MostDetailedMip](#) on down to least detailed.

## Remarks

This structure is one member of a shader-resource-view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_SRV1 structure (d3d11\_3.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2D_SRV1 {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT PlaneSlice;
} D3D11_TEX2D_SRV1;
```

## Members

`MostDetailedMip`

Index of the most detailed mipmap level to use; this number is between 0 and `(MipLevels` (from the original Texture2D for which `ID3D11Device3::CreateShaderResourceView1` creates a view) - 1 ).

`MipLevels`

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D11\\_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.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_TEX2D\_UAV structure (d3d11.h)

Article 02/22/2024

Describes a unordered-access 2D texture resource.

## Syntax

C++

```
typedef struct D3D11_TEX2D_UAV {
    UINT MipSlice;
} D3D11_TEX2D_UAV;
```

## Members

MipSlice

Type: [UINT](#)

The mipmap slice index.

## Remarks

This structure is used by a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2D\_UAV1 structure (d3d11\_3.h)

Article 02/22/2024

Describes a unordered-access 2D texture resource.

## Syntax

C++

```
typedef struct D3D11_TEX2D_UAV1 {
    UINT MipSlice;
    UINT PlaneSlice;
} D3D11_TEX2D_UAV1;
```

## Members

`MipSlice`

The mipmap slice index.

`PlaneSlice`

The index (plane slice number) of the plane to use in the texture.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2DMS\_ARRAY\_DSV structure (d3d11.h)

Article 02/22/2024

Specifies the subresources from an array of multisampled 2D textures for a depth-stencil view.

## Syntax

C++

```
typedef struct D3D11_TEX2DMS_ARRAY_DSV {
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX2DMS_ARRAY_DSV;
```

## Members

`FirstArraySlice`

Type: [UINT](#)

The index of the first texture to use in an array of textures.

`ArraySize`

Type: [UINT](#)

Number of textures to use.

## Remarks

This structure is one member of a depth-stencil-view description (see [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2DMS\_ARRAY\_RTV structure (d3d11.h)

Article 02/22/2024

Specifies the subresources from a an array of multisampled 2D textures to use in a render-target view.

## Syntax

C++

```
typedef struct D3D11_TEX2DMS_ARRAY_RTV {
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX2DMS_ARRAY_RTV;
```

## Members

`FirstArraySlice`

Type: [UINT](#)

The index of the first texture to use in an array of textures.

`ArraySize`

Type: [UINT](#)

Number of textures to use.

## Remarks

This structure is one member of a render-target-view description (see [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2DMS\_ARRAY\_SRV structure (d3d11.h)

Article 02/22/2024

Specifies the subresources from an array of multisampled 2D textures to use in a shader-resource view.

## Syntax

C++

```
typedef struct D3D11_TEX2DMS_ARRAY_SRV {
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D11_TEX2DMS_ARRAY_SRV;
```

## Members

`FirstArraySlice`

Type: [UINT](#)

The index of the first texture to use in an array of textures.

`ArraySize`

Type: [UINT](#)

Number of textures to use.

## Remarks

This structure is one member of a shader-resource-view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2DMS\_DSV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX2DMS_DSV {
    UINT UnusedField_NothingToDefine;
} D3D11_TEX2DMS_DSV;
```

## Members

UnusedField\_NothingToDefine

Type: [UINT](#)

Unused.

## Remarks

Because a multisampled 2D texture contains a single subtexture, there is nothing to specify; this unused member is included so that this structure will compile in C.

## Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2DMS\_RTV structure (d3d11.h)

Article 02/22/2024

Specifies the subresource from a multisampled 2D texture to use in a render-target view.

## Syntax

C++

```
typedef struct D3D11_TEX2DMS_RTV {
    UINT UnusedField_NothingToDefine;
} D3D11_TEX2DMS_RTV;
```

## Members

UnusedField\_NothingToDefine

Type: [UINT](#)

Integer of any value. See remarks.

## Remarks

Since a multisampled 2D texture contains a single subresource, there is actually nothing to specify in D3D11\_TEX2DMS\_RTV. Consequently, **UnusedField\_NothingToDefine** is included so that this structure will compile in C.

## Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX2DMS\_SRV structure (d3d11.h)

Article 02/22/2024

Specifies the subresources from a multisampled 2D texture to use in a shader-resource view.

## Syntax

C++

```
typedef struct D3D11_TEX2DMS_SRV {
    UINT UnusedField_NothingToDefine;
} D3D11_TEX2DMS_SRV;
```

## Members

UnusedField\_NothingToDefine

Type: [UINT](#)

Integer of any value. See remarks.

## Remarks

Since a multisampled 2D texture contains a single subresource, there is actually nothing to specify in D3D11\_TEX2DMS\_SRV. Consequently, **UnusedField\_NothingToDefine** is included so that this structure will compile in C.

## Requirements

[Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX3D\_RTV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX3D_RTV {
    UINT MipSlice;
    UINT FirstWSlice;
    UINT WSize;
} D3D11_TEX3D_RTV;
```

## Members

MipSlice

Type: [UINT](#)

The index of the mipmap level to use mip slice.

FirstWSlice

Type: [UINT](#)

First depth level to use.

WSize

Type: [UINT](#)

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

This structure is one member of a render target view. See

[D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#).

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX3D\_SRV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEX3D_SRV {  
    UINT MostDetailedMip;  
    UINT MipLevels;  
} D3D11_TEX3D_SRV;
```

## Members

`MostDetailedMip`

Type: [UINT](#)

Index of the most detailed mipmap level to use; this number is between 0 and [MipLevels](#) (from the original Texture3D for which [ID3D11Device::CreateShaderResourceView](#) creates a view) -1.

`MipLevels`

Type: [UINT](#)

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#).

Set to -1 to indicate all the mipmap levels from [MostDetailedMip](#) on down to least detailed.

## Remarks

This structure is one member of a shader-resource-view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEX3D\_UAV structure (d3d11.h)

Article 02/22/2024

Describes a unordered-access 3D texture resource.

## Syntax

C++

```
typedef struct D3D11_TEX3D_UAV {
    UINT MipSlice;
    UINT FirstWSlice;
    UINT WSize;
} D3D11_TEX3D_UAV;
```

## Members

MipSlice

Type: **UINT**

The mipmap slice index.

FirstWSlice

Type: **UINT**

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

WSize

Type: **UINT**

The number of depth slices.

## Remarks

This structure is used by a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEXCUBE\_ARRAY\_SRV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEXCUBE_ARRAY_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT First2DArrayFace;
    UINT NumCubes;
} D3D11_TEXCUBE_ARRAY_SRV;
```

## Members

**MostDetailedMip**

Type: [UINT](#)

Index of the most detailed mipmap level to use; this number is between 0 and [MipLevels](#) (from the original TextureCube for which [ID3D11Device::CreateShaderResourceView](#) creates a view) -1.

**MipLevels**

Type: [UINT](#)

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#).

Set to -1 to indicate all the mipmap levels from **MostDetailedMip** on down to least detailed.

**First2DArrayFace**

Type: [UINT](#)

Index of the first 2D texture to use.

NumCubes

Type: [UINT](#)

Number of cube textures in the array.

## Remarks

This structure is one member of a shader-resource-view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)).

## Requirements

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

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEXCUBE\_SRV structure (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef struct D3D11_TEXCUBE_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
} D3D11_TEXCUBE_SRV;
```

## Members

`MostDetailedMip`

Type: [UINT](#)

Index of the most detailed mipmap level to use; this number is between 0 and [MipLevels](#) (from the original TextureCube for which [ID3D11Device::CreateShaderResourceView](#) creates a view) -1.

`MipLevels`

Type: [UINT](#)

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#).

Set to -1 to indicate all the mipmap levels from [MostDetailedMip](#) on down to least detailed.

## Remarks

This structure is one member of a shader-resource-view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)).

## Requirements

Expand table

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEXTURE1D\_DESC structure (d3d11.h)

Article07/27/2022

Describes a 1D texture.

## Syntax

C++

```
typedef struct D3D11_TEXTURE1D_DESC {
    UINT          Width;
    UINT          MipLevels;
    UINT          ArraySize;
    DXGI_FORMAT   Format;
    D3D11_USAGE   Usage;
    UINT          BindFlags;
    UINT          CPUAccessFlags;
    UINT          MiscFlags;
} D3D11_TEXTURE1D_DESC;
```

## Members

Width

Type: [UINT](#)

Texture width (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE1D\_U\_DIMENSION (16384). However, the range is actually constrained by the [feature level](#) at which you create the rendering device. For more information about restrictions, see Remarks.

MipLevels

Type: [UINT](#)

The maximum number of mipmap levels in the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#). Use 1 for a multisampled texture; or 0 to generate a full set of subtextures.

ArraySize

Type: [UINT](#)

Number of textures in the array. The range is from 1 to D3D11\_REQ\_TEXTURE1D\_ARRAY\_AXIS\_DIMENSION (2048). However, the range is actually constrained by the [feature level](#) at which you create the rendering device. For more information about restrictions, see Remarks.

#### Format

Type: [DXGI\\_FORMAT](#)

Texture format (see [DXGI\\_FORMAT](#)).

#### Usage

Type: [D3D11\\_USAGE](#)

Value that identifies how the texture is to be read from and written to. The most common value is D3D11\_USAGE\_DEFAULT; see [D3D11\\_USAGE](#) for all possible values.

#### BindFlags

Type: [UINT](#)

Flags (see [D3D11\\_BIND\\_FLAG](#)) for binding to pipeline stages. The flags can be combined by a bitwise OR. For a 1D texture, the allowable values are:  
D3D11\_BIND\_SHADER\_RESOURCE, D3D11\_BIND\_RENDER\_TARGET and  
D3D11\_BIND\_DEPTH\_STENCIL.

#### CPUAccessFlags

Type: [UINT](#)

Flags (see [D3D11\\_CPU\\_ACCESS\\_FLAG](#)) to specify the types of CPU access allowed. Use 0 if CPU access is not required. These flags can be combined with a bitwise OR.

#### MiscFlags

Type: [UINT](#)

Flags (see [D3D11\\_RESOURCE\\_MISC\\_FLAG](#)) that identify other, less common resource options. Use 0 if none of these flags apply. These flags can be combined with a bitwise OR.

## Remarks

This structure is used in a call to [ID3D11Device::CreateTexture1D](#).

In addition to this structure, you can also use the [CD3D11\\_TEXTURE1D\\_DESC](#) derived structure, which is defined in D3D11.h and behaves like an inherited class, to help create a texture description.

The texture size range is determined by the [feature level](#) at which you create the device and not the Microsoft Direct3D interface version. For example, if you use Microsoft Direct3D 10 hardware at feature level 10 ([D3D\\_FEATURE\\_LEVEL\\_10\\_0](#)) and call [D3D11CreateDevice](#) to create an [ID3D11Device](#), you must constrain the maximum texture size to [D3D10\\_REQ\\_TEXTURE1D\\_U\\_DIMENSION](#) (8192) when you create your 1D texture.

## Requirements

Header	d3d11.h
--------	---------

## See also

[Resource Structures](#)

## Feedback

Was this page helpful?



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

# D3D11\_TEXTURE2D\_DESC structure (d3d11.h)

Article 07/27/2022

Describes a 2D texture.

## Syntax

C++

```
typedef struct D3D11_TEXTURE2D_DESC {
    UINT           Width;
    UINT           Height;
    UINT           MipLevels;
    UINT           ArraySize;
    DXGI_FORMAT    Format;
    DXGI_SAMPLE_DESC SampleDesc;
    D3D11_USAGE    Usage;
    UINT           BindFlags;
    UINT           CPUAccessFlags;
    UINT           MiscFlags;
} D3D11_TEXTURE2D_DESC;
```

## Members

Width

Type: [UINT](#)

Texture width (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE2D\_U\_OR\_V\_DIMENSION (16384). For a texture cube-map, the range is from 1 to D3D11\_REQ\_TEXTURECUBE\_DIMENSION (16384). However, the range is actually constrained by the [feature level](#) at which you create the rendering device. For more information about restrictions, see Remarks.

Height

Type: [UINT](#)

Texture height (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE2D\_U\_OR\_V\_DIMENSION (16384). For a texture cube-map, the range is from 1 to D3D11\_REQ\_TEXTURECUBE\_DIMENSION (16384). However, the range

is actually constrained by the [feature level](#) at which you create the rendering device. For more information about restrictions, see Remarks.

#### MipLevels

Type: [UINT](#)

The maximum number of mipmap levels in the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#). Use 1 for a multisampled texture; or 0 to generate a full set of subtextures.

#### ArraySize

Type: [UINT](#)

Number of textures in the texture array. The range is from 1 to [D3D11\\_REQ\\_TEXTURE2D\\_ARRAY\\_AXIS\\_DIMENSION](#) (2048). For a texture cube-map, this value is a multiple of 6 (that is, 6 times the value in the **NumCubes** member of [D3D11\\_TEXCUBE\\_ARRAY\\_SRV](#)), and the range is from 6 to 2046. The range is actually constrained by the [feature level](#) at which you create the rendering device. For more information about restrictions, see Remarks.

#### Format

Type: [DXGI\\_FORMAT](#)

Texture format (see [DXGI\\_FORMAT](#)).

#### SampleDesc

Type: [DXGI\\_SAMPLE\\_DESC](#)

Structure that specifies multisampling parameters for the texture. See [DXGI\\_SAMPLE\\_DESC](#).

#### Usage

Type: [D3D11\\_USAGE](#)

Value that identifies how the texture is to be read from and written to. The most common value is [D3D11\\_USAGE\\_DEFAULT](#); see [D3D11\\_USAGE](#) for all possible values.

#### BindFlags

Type: [UINT](#)

Flags (see [D3D11\\_BIND\\_FLAG](#)) for binding to pipeline stages. The flags can be combined by a bitwise OR.

#### CPUAccessFlags

Type: [UINT](#)

Flags (see [D3D11\\_CPU\\_ACCESS\\_FLAG](#)) to specify the types of CPU access allowed. Use 0 if CPU access is not required. These flags can be combined with a bitwise OR.

#### MiscFlags

Type: [UINT](#)

Flags (see [D3D11\\_RESOURCE\\_MISC\\_FLAG](#)) that identify other, less common resource options. Use 0 if none of these flags apply. These flags can be combined by using a bitwise OR. For a texture cube-map, set the [D3D11\\_RESOURCE\\_MISC\\_TEXTURECUBE](#) flag. Cube-map arrays (that is, [ArraySize > 6](#)) require feature level [D3D\\_FEATURE\\_LEVEL\\_10\\_1](#) or higher.

## Remarks

This structure is used in a call to [ID3D11Device::CreateTexture2D](#).

In addition to this structure, you can also use the [CD3D11\\_TEXTURE2D\\_DESC](#) derived structure, which is defined in D3D11.h and behaves like an inherited class, to help create a texture description.

The device places some size restrictions (must be multiples of a minimum size) for a subsampled, block compressed, or bit-format resource.

The texture size range is determined by the [feature level](#) at which you create the device and not the Microsoft Direct3D interface version. For example, if you use Microsoft Direct3D 10 hardware at feature level 10 ([D3D\\_FEATURE\\_LEVEL\\_10\\_0](#)) and call [D3D11CreateDevice](#) to create an [ID3D11Device](#), you must constrain the maximum texture size to [D3D10\\_REQ\\_TEXTURE2D\\_U\\_OR\\_V\\_DIMENSION](#) (8192) when you create your 2D texture.

## Requirements

Header	d3d11.h
--------	---------

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# CD3D11\_TEXTURE2D\_DESC1 structure (d3d11\_3.h)

Article 02/22/2024

Describes a 2D texture.

## Syntax

C++

```
struct CD3D11_TEXTURE2D_DESC1 : D3D11_TEXTURE2D_DESC1 {
    void CD3D11_TEXTURE2D_DESC1();
    void CD3D11_TEXTURE2D_DESC1(
        const D3D11_TEXTURE2D_DESC1 & o
    );
    void CD3D11_TEXTURE2D_DESC1(
        DXGI_FORMAT           format,
        UINT                  width,
        UINT                  height,
        UINT                  arraySize,
        UINT                  mipLevels,
        UINT                  bindFlags,
        D3D11_USAGE           usage,
        UINT                  cpuaccessFlags,
        UINT                  sampleCount,
        UINT                  sampleQuality,
        UINT                  miscFlags,
        D3D11_TEXTURE_LAYOUT  textureLayout
    );
    void CD3D11_TEXTURE2D_DESC1(
        const D3D11_TEXTURE2D_DESC & desc,
        D3D11_TEXTURE_LAYOUT       textureLayout
    );
    void ~CD3D11_TEXTURE2D_DESC1();
};
```

## Inheritance

The `CD3D11_TEXTURE2D_DESC1` structure implements `D3D11_TEXTURE2D_DESC1`.

## Members

```
void CD3D11_TEXTURE2D_DESC1()
```

TBD

```
void CD3D11_TEXTURE2D_DESC1( const D3D11_TEXTURE2D_DESC1 & o)

void CD3D11_TEXTURE2D_DESC1( DXGI_FORMAT format, UINT width, UINT height, UINT
arraySize, UINT mipLevels, UINT bindFlags, D3D11_USAGE usage, UINT cpuaccessFlags,
UINT sampleCount, UINT sampleQuality, UINT miscFlags, D3D11_TEXTURE_LAYOUT
textureLayout)

void CD3D11_TEXTURE2D_DESC1( const D3D11_TEXTURE2D_DESC & desc,
D3D11_TEXTURE_LAYOUT textureLayout)

void ~CD3D11_TEXTURE2D_DESC1()
```

TBD

## Remarks

This structure is used in a call to [ID3D11Device3::CreateTexture2D1](#).

In addition to this structure, you can also use the **CD3D11\_TEXTURE2D\_DESC1** derived structure, which is defined in D3D11\_3.h and behaves like an inherited class, to help create a texture description.

The device places some size restrictions (must be multiples of a minimum size) for a subsampled, block compressed, or bit-format resource.

The texture size range is determined by the [feature level](#) at which you create the device and not the Microsoft Direct3D interface version. For example, if you use Microsoft Direct3D 10 hardware at feature level 10 ([D3D\\_FEATURE\\_LEVEL\\_10\\_0](#)) and call [D3D11CreateDevice](#) to create an [ID3D11Device](#), you must constrain the maximum texture size to D3D10\_REQ\_TEXTURE2D\_U\_OR\_V\_DIMENSION (8192) when you create your 2D texture.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TEXTURE3D\_DESC structure (d3d11.h)

Article 07/27/2022

Describes a 3D texture.

## Syntax

C++

```
typedef struct D3D11_TEXTURE3D_DESC {
    UINT          Width;
    UINT          Height;
    UINT          Depth;
    UINT          MipLevels;
    DXGI_FORMAT   Format;
    D3D11_USAGE   Usage;
    UINT          BindFlags;
    UINT          CPUAccessFlags;
    UINT          MiscFlags;
} D3D11_TEXTURE3D_DESC;
```

## Members

Width

Type: [UINT](#)

Texture width (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE3D\_U\_V\_OR\_W\_DIMENSION (2048). However, the range is actually constrained by the [feature level](#) at which you create the rendering device. For more information about restrictions, see Remarks.

Height

Type: [UINT](#)

Texture height (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE3D\_U\_V\_OR\_W\_DIMENSION (2048). However, the range is actually constrained by the [feature level](#) at which you create the rendering device. For more information about restrictions, see Remarks.

### Depth

Type: [UINT](#)

Texture depth (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE3D\_U\_V\_OR\_W\_DIMENSION (2048). However, the range is actually constrained by the [feature level](#) at which you create the rendering device. For more information about restrictions, see Remarks.

### MipLevels

Type: [UINT](#)

The maximum number of mipmap levels in the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#). Use 1 for a multisampled texture; or 0 to generate a full set of subtextures.

### Format

Type: [DXGI\\_FORMAT](#)

Texture format (see [DXGI\\_FORMAT](#)).

### Usage

Type: [D3D11\\_USAGE](#)

Value that identifies how the texture is to be read from and written to. The most common value is D3D11\_USAGE\_DEFAULT; see [D3D11\\_USAGE](#) for all possible values.

### BindFlags

Type: [UINT](#)

Flags (see [D3D11\\_BIND\\_FLAG](#)) for binding to pipeline stages. The flags can be combined by a bitwise OR.

### CPUAccessFlags

Type: [UINT](#)

Flags (see [D3D11\\_CPU\\_ACCESS\\_FLAG](#)) to specify the types of CPU access allowed. Use 0 if CPU access is not required. These flags can be combined with a bitwise OR.

### MiscFlags

Type: [UINT](#)

Flags (see [D3D11\\_RESOURCE\\_MISC\\_FLAG](#)) that identify other, less common resource options. Use 0 if none of these flags apply. These flags can be combined with a bitwise OR.

## Remarks

This structure is used in a call to [ID3D11Device::CreateTexture3D](#).

In addition to this structure, you can also use the [CD3D11\\_TEXTURE3D\\_DESC](#) derived structure, which is defined in D3D11.h and behaves like an inherited class, to help create a texture description.

The device restricts the size of subsampled, block compressed, and bit format resources to be multiples of sizes specific to each format.

The texture size range is determined by the [feature level](#) at which you create the device and not the Microsoft Direct3D interface version. For example, if you use Microsoft Direct3D 10 hardware at feature level 10 ([D3D\\_FEATURE\\_LEVEL\\_10\\_0](#)) and call [D3D11CreateDevice](#) to create an [ID3D11Device](#), you must constrain the maximum texture size to D3D10\_REQ\_TEXTURE3D\_U\_V\_OR\_W\_DIMENSION (2048) when you create your 3D texture.

## Requirements

Header	d3d11.h
--------	---------

## See also

[Resource Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# CD3D11\_TEXTURE3D\_DESC1 structure (d3d11\_3.h)

Article 02/22/2024

Describes a 3D texture.

## Syntax

C++

```
struct CD3D11_TEXTURE3D_DESC1 : D3D11_TEXTURE3D_DESC1 {
    void CD3D11_TEXTURE3D_DESC1();
    void CD3D11_TEXTURE3D_DESC1(
        const D3D11_TEXTURE3D_DESC1 & o
    );
    void CD3D11_TEXTURE3D_DESC1(
        DXGI_FORMAT             format,
        UINT                   width,
        UINT                   height,
        UINT                   depth,
        UINT                   mipLevels,
        UINT                   bindFlags,
        D3D11_USAGE            usage,
        UINT                   cpuAccessFlags,
        UINT                   miscFlags,
        D3D11_TEXTURE_LAYOUT   textureLayout
    );
    void CD3D11_TEXTURE3D_DESC1(
        const D3D11_TEXTURE3D_DESC & desc,
        D3D11_TEXTURE_LAYOUT      textureLayout
    );
    void ~CD3D11_TEXTURE3D_DESC1();
};
```

## Inheritance

The CD3D11\_TEXTURE3D\_DESC1 structure implements D3D11\_TEXTURE3D\_DESC1.

## Members

```
void CD3D11_TEXTURE3D_DESC1()
```

TBD

```

void CD3D11_TEXTURE3D_DESC1( const D3D11_TEXTURE3D_DESC1 & o)

void CD3D11_TEXTURE3D_DESC1( DXGI_FORMAT format, UINT width, UINT height, UINT
depth, UINT mipLevels, UINT bindFlags, D3D11_USAGE usage, UINT cpuaccessFlags, UINT
miscFlags, D3D11_TEXTURE_LAYOUT textureLayout)

void CD3D11_TEXTURE3D_DESC1( const D3D11_TEXTURE3D_DESC & desc,
D3D11_TEXTURE_LAYOUT textureLayout)

void ~CD3D11_TEXTURE3D_DESC1()

```

TBD

## Remarks

This structure is used in a call to [ID3D11Device3::CreateTexture3D1](#).

In addition to this structure, you can also use the **CD3D11\_TEXTURE3D\_DESC1** derived structure, which is defined in D3D11\_3.h and behaves like an inherited class, to help create a texture description.

The device restricts the size of subsampled, block compressed, and bit format resources to be multiples of sizes specific to each format.

The texture size range is determined by the [feature level](#) at which you create the device and not the Microsoft Direct3D interface version. For example, if you use Microsoft Direct3D 10 hardware at feature level 10 ([D3D\\_FEATURE\\_LEVEL\\_10\\_0](#)) and call [D3D11CreateDevice](#) to create an [ID3D11Device](#), you must constrain the maximum texture size to [D3D10\\_REQ\\_TEXTURE3D\\_U\\_V\\_OR\\_W\\_DIMENSION](#) (2048) when you create your 3D texture.

## Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TILE\_REGION\_SIZE structure (d3d11\_2.h)

Article 02/22/2024

Describes the size of a tiled region.

## Syntax

C++

```
typedef struct D3D11_TILE_REGION_SIZE {
    UINT    NumTiles;
    BOOL    bUseBox;
    UINT    Width;
    UINT16  Height;
    UINT16  Depth;
} D3D11_TILE_REGION_SIZE;
```

## Members

NumTiles

Type: **UINT**

The number of tiles in the tiled region.

bUseBox

Type: **BOOL**

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.

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 **bUseBox**, you use a **D3D11\_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, **bUseBox** 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

Type: **UINT**

The width of the tiled region, in tiles. Used for buffer and 1D, 2D, and 3D textures.

#### Height

Type: **UINT16**

The height of the tiled region, in tiles. Used for 2D and 3D textures.

#### Depth

Type: **UINT16**

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.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11_2.h

## See also

[D3D11\\_TILED\\_RESOURCE\\_COORDINATE](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TILED\_RESOURCE\_COORDINATE structure (d3d11\_2.h)

Article 02/22/2024

Describes the coordinates of a tiled resource.

## Syntax

C++

```
typedef struct D3D11_TILED_RESOURCE_COORDINATE {
    UINT X;
    UINT Y;
    UINT Z;
    UINT Subresource;
} D3D11_TILED_RESOURCE_COORDINATE;
```

## Members

X

Type: [UINT](#)

The x position of a tiled resource. Used for buffer and 1D, 2D, and 3D textures.

Y

Type: [UINT](#)

The y position of a tiled resource. Used for 2D and 3D textures.

Z

Type: [UINT](#)

The z position of a tiled resource. Used for 3D textures.

Subresource

Type: [UINT](#)

A subresource index value into mipmaps and arrays. Used for 1D, 2D, and 3D textures.

For mipmaps that use nonstandard tiling, or are packed, or both use nonstandard tiling and are packed, any subresource value that indicates any of the packed mipmaps all refer to the same tile.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11_2.h

## See also

[Resource Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TILE\_SHAPE structure (d3d11\_2.h)

Article 02/22/2024

Describes the shape of a tile by specifying its dimensions.

## Syntax

C++

```
typedef struct D3D11_TILE_SHAPE {
    UINT WidthInTexels;
    UINT HeightInTexels;
    UINT DepthInTexels;
} D3D11_TILE_SHAPE;
```

## Members

`WidthInTexels`

Type: **UINT**

The width in texels of the tile.

`HeightInTexels`

Type: **UINT**

The height in texels of the tile.

`DepthInTexels`

Type: **UINT**

The depth in texels of the tile.

## Remarks

Texels are equivalent to pixels. For untyped buffer resources, a texel is just a byte. For multisample antialiasing (MSAA) surfaces, the numbers are still in terms of pixels/texels. The values here are independent of the surface dimensions. Even if the surface is smaller than what would fit in a tile, the full tile dimensions are reported here.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11_2.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_UNORDERED\_ACCESS\_VIEW\_DESC

## C structure (d3d11.h)

Article 02/22/2024

Specifies the subresources from a resource that are accessible using an unordered-access view.

## Syntax

C++

```
typedef struct D3D11_UNORDERED_ACCESS_VIEW_DESC {
    DXGI_FORMAT           Format;
    D3D11_UAV_DIMENSION ViewDimension;
    union {
        D3D11_BUFFER_UAV     Buffer;
        D3D11_TEX1D_UAV      Texture1D;
        D3D11_TEX1D_ARRAY_UAV Texture1DArray;
        D3D11_TEX2D_UAV      Texture2D;
        D3D11_TEX2D_ARRAY_UAV Texture2DArray;
        D3D11_TEX3D_UAV      Texture3D;
    };
} D3D11_UNORDERED_ACCESS_VIEW_DESC;
```

## Members

Format

Type: [DXGI\\_FORMAT](#)

The data format (see [DXGI\\_FORMAT](#)).

ViewDimension

Type: [D3D11\\_UAV\\_DIMENSION](#)

The resource type (see [D3D11\\_UAV\\_DIMENSION](#)), which specifies how the resource will be accessed.

Buffer

Type: [D3D11\\_BUFFER\\_UAV](#)

Specifies which buffer elements can be accessed (see [D3D11\\_BUFFER\\_UAV](#)).

#### Texture1D

Type: [D3D11\\_TEX1D\\_UAV](#)

Specifies the subresources in a 1D texture that can be accessed (see [D3D11\\_TEX1D\\_UAV](#)).

#### Texture1DArray

Type: [D3D11\\_TEX1D\\_ARRAY\\_UAV](#)

Specifies the subresources in a 1D texture array that can be accessed (see [D3D11\\_TEX1D\\_ARRAY\\_UAV](#)).

#### Texture2D

Type: [D3D11\\_TEX2D\\_UAV](#)

Specifies the subresources in a 2D texture that can be accessed (see [D3D11\\_TEX2D\\_UAV](#)).

#### Texture2DArray

Type: [D3D11\\_TEX2D\\_ARRAY\\_UAV](#)

Specifies the subresources in a 2D texture array that can be accessed (see [D3D11\\_TEX2D\\_ARRAY\\_UAV](#)).

#### Texture3D

Type: [D3D11\\_TEX3D\\_UAV](#)

Specifies subresources in a 3D texture that can be accessed (see [D3D11\\_TEX3D\\_UAV](#)).

## Remarks

An unordered-access-view description is passed into [ID3D11Device::CreateUnorderedAccessView](#) to create a view.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC1 structure (d3d11\_3.h)

Article02/22/2024

Describes the subresources from a resource that are accessible using an unordered-access view.

## Syntax

C++

```
struct CD3D11_UNORDERED_ACCESS_VIEW_DESC1 :  
D3D11_UNORDERED_ACCESS_VIEW_DESC1 {  
    void CD3D11_UNORDERED_ACCESS_VIEW_DESC1();  
    void CD3D11_UNORDERED_ACCESS_VIEW_DESC1(  
        const D3D11_UNORDERED_ACCESS_VIEW_DESC1 & o  
    );  
    void CD3D11_UNORDERED_ACCESS_VIEW_DESC1(  
        D3D11_UAV_DIMENSION viewDimension,  
        DXGI_FORMAT format,  
        UINT mipSlice,  
        UINT firstArraySlice,  
        UINT arraySize,  
        UINT flags,  
        UINT planeSlice  
    );  
    void CD3D11_UNORDERED_ACCESS_VIEW_DESC1(  
        ID3D11Buffer *unnamedParam1,  
        DXGI_FORMAT format,  
        UINT firstElement,  
        UINT numElements,  
        UINT flags  
    );  
    void CD3D11_UNORDERED_ACCESS_VIEW_DESC1(  
        ID3D11Texture1D *pTex1D,  
        D3D11_UAV_DIMENSION viewDimension,  
        DXGI_FORMAT format,  
        UINT mipSlice,  
        UINT firstArraySlice,  
        UINT arraySize  
    );  
    void CD3D11_UNORDERED_ACCESS_VIEW_DESC1(  
        ID3D11Texture2D *pTex2D,  
        D3D11_UAV_DIMENSION viewDimension,  
        DXGI_FORMAT format,  
        UINT mipSlice,  
        UINT firstArraySlice,  
        UINT arraySize,
```

```

    UINT          planeSlice
);
void CD3D11_UNORDERED_ACCESS_VIEW_DESC1(
    ID3D11Texture3D *pTex3D,
    DXGI_FORMAT      format,
    UINT             mipSlice,
    UINT             firstWSlice,
    UINT             wSize
);
void ~CD3D11_UNORDERED_ACCESS_VIEW_DESC1();
};

```

## Inheritance

The **CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC1** structure implements **D3D11\_UNORDERED\_ACCESS\_VIEW\_DESC1**.

## Members

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC1()
```

TBD

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC1( const D3D11_UNORDERED_ACCESS_VIEW_DESC1 &
o)
```

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC1( D3D11_UAV_DIMENSION viewDimension,
DXGI_FORMAT format, UINT mipSlice, UINT firstArraySlice, UINT arraySize, UINT
flags, UINT planeSlice)
```

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC1( ID3D11Buffer *unnamedParam1, DXGI_FORMAT
format, UINT firstElement, UINT numElements, UINT flags)
```

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC1( ID3D11Texture1D *pTex1D,
D3D11_UAV_DIMENSION viewDimension, DXGI_FORMAT format, UINT mipSlice, UINT
firstArraySlice, UINT arraySize)
```

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC1( ID3D11Texture2D *pTex2D,
D3D11_UAV_DIMENSION viewDimension, DXGI_FORMAT format, UINT mipSlice, UINT
firstArraySlice, UINT arraySize, UINT planeSlice)
```

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC1( ID3D11Texture3D *pTex3D, DXGI_FORMAT
format, UINT mipSlice, UINT firstWSlice, UINT wSize)
```

```
void ~CD3D11_UNORDERED_ACCESS_VIEW_DESC1()
```

TBD

## Remarks

An unordered-access-view description is passed into [ID3D11Device3::CreateUnorderedAccessView1](#) to create a view.

## Requirements

[Expand table](#)

Requirement	Value
Header	d3d11_3.h

## See also

[Resource Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Resource Enumerations (Direct3D 11 Graphics)

Article • 12/10/2020

Enumerations are used to specify information about how resources are created and accessed during rendering.

## In this section

Topic	Description
<a href="#">D3D11_BIND_FLAG</a>	Identifies how to bind a resource to the pipeline.
<a href="#">D3D11_BUFFEREX_SRV_FLAG</a>	Identifies how to view a buffer resource.
<a href="#">D3D11_BUFFER_UAV_FLAG</a>	Identifies unordered-access view options for a buffer resource.
<a href="#">D3D11_CHECK_MULTISAMPLE_QUALITY_LEVELS_FLAG</a>	Identifies how to check multisample quality levels.
<a href="#">D3D11_CPU_ACCESS_FLAG</a>	Specifies the types of CPU access allowed for a resource.
<a href="#">D3D11_DSV_DIMENSION</a>	Specifies how to access a resource used in a depth-stencil view.
<a href="#">D3D11_DSV_FLAG</a>	Depth-stencil view options.
<a href="#">D3D11_MAP</a>	Identifies a resource to be accessed for reading and writing by the CPU. Applications may combine one or more of these flags.
<a href="#">D3D11_MAP_FLAG</a>	Specifies how the CPU should respond when an application calls the <a href="#">ID3D11DeviceContext::Map</a> method on a resource that is being used by the GPU.
<a href="#">D3D11_RESOURCE_DIMENSION</a>	Identifies the type of resource being used.
<a href="#">D3D11_RESOURCE_MISC_FLAG</a>	Identifies options for resources.

Topic	Description
<a href="#">D3D11_RTV_DIMENSION</a>	These flags identify the type of resource that will be viewed as a render target.
<a href="#">D3D11_SRV_DIMENSION</a>	These flags identify the type of resource that will be viewed as a shader resource.
<a href="#">D3D11_STANDARD_MULTISAMPLE_QUALITY_LEVELS</a>	Specifies a multi-sample pattern type.
<a href="#">D3D11_TEXTURE_LAYOUT</a>	Specifies texture layout options.
<a href="#">D3D11_TILE_COPY_FLAG</a>	Identifies how to copy a tile.
<a href="#">D3D11_TILE_MAPPING_FLAG</a>	Identifies how to perform a tile-mapping operation.
<a href="#">D3D11_TILE_RANGE_FLAG</a>	Specifies a range of tile mappings to use with <a href="#">ID3D11DeviceContext2::UpdateTiles</a> .
<a href="#">D3D11_UAV_DIMENSION</a>	Unordered-access view options.
<a href="#">D3D11_USAGE</a>	Identifies expected resource use during rendering. The usage directly reflects whether a resource is accessible by the CPU and/or the graphics processing unit (GPU).

## Related topics

[Resource Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BIND\_FLAG enumeration (d3d11.h)

Article 07/27/2022

Identifies how to bind a resource to the pipeline.

## Syntax

C++

```
typedef enum D3D11_BIND_FLAG {
    D3D11_BIND_VERTEX_BUFFER = 0x1L,
    D3D11_BIND_INDEX_BUFFER = 0x2L,
    D3D11_BIND_CONSTANT_BUFFER = 0x4L,
    D3D11_BIND_SHADER_RESOURCE = 0x8L,
    D3D11_BIND_STREAM_OUTPUT = 0x10L,
    D3D11_BIND_RENDER_TARGET = 0x20L,
    D3D11_BIND_DEPTH_STENCIL = 0x40L,
    D3D11_BIND_UNORDERED_ACCESS = 0x80L,
    D3D11_BIND_DECODER = 0x200L,
    D3D11_BIND_VIDEO_ENCODER = 0x400L
};
```

## Constants

D3D11\_BIND\_VERTEX\_BUFFER

Value: *0x1L*

Bind a buffer as a vertex buffer to the input-assembler stage.

D3D11\_BIND\_INDEX\_BUFFER

Value: *0x2L*

Bind a buffer as an index buffer to the input-assembler stage.

D3D11\_BIND\_CONSTANT\_BUFFER

Value: *0x4L*

Bind a buffer as a constant buffer to a shader stage; this flag may NOT be combined with any other bind flag.

**D3D11\_BIND\_SHADER\_RESOURCE**

Value: *0x8L*

Bind a buffer or texture to a shader stage; this flag cannot be used with the [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#) flag.

**Note** The Direct3D 11.1 runtime, which is available starting with Windows 8, enables mapping dynamic constant buffers and shader resource views (SRVs) of dynamic buffers with [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#). The Direct3D 11 and earlier runtimes limited mapping to vertex or index buffers. To determine if a Direct3D device supports these features, call [ID3D11Device::CheckFeatureSupport](#) with [D3D11\\_FEATURE\\_D3D11\\_OPTIONS](#). [CheckFeatureSupport](#) fills members of a [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) structure with the device's features. The relevant members here are [MapNoOverwriteOnDynamicConstantBuffer](#) and [MapNoOverwriteOnDynamicBufferSRV](#).

**D3D11\_BIND\_STREAM\_OUTPUT**

Value: *0x10L*

Bind an output buffer for the stream-output stage.

**D3D11\_BIND\_RENDER\_TARGET**

Value: *0x20L*

Bind a texture as a render target for the output-merger stage.

**D3D11\_BIND\_DEPTH\_STENCIL**

Value: *0x40L*

Bind a texture as a depth-stencil target for the output-merger stage.

**D3D11\_BIND\_UNORDERED\_ACCESS**

Value: *0x80L*

Bind an [unordered access](#) resource.

**D3D11\_BIND\_DECODER**

Value: *0x200L*

Set this flag to indicate that a [2D texture](#) is used to receive output from the decoder API. The common way to create resources for a decoder output is by calling the [ID3D11Device::CreateTexture2D](#) method to create an array of 2D textures. However, you cannot use texture arrays that are created with this flag in calls to [ID3D11Device::CreateShaderResourceView](#).

**Direct3D 11:** This value is not supported until Direct3D 11.1.

#### D3D11\_BIND\_VIDEO\_ENCODER

Value: 0x400L

Set this flag to indicate that a [2D texture](#) is used to receive input from the video encoder API. The common way to create resources for a video encoder is by calling the [ID3D11Device::CreateTexture2D](#) method to create an array of 2D textures. However, you cannot use texture arrays that are created with this flag in calls to [ID3D11Device::CreateShaderResourceView](#).

**Direct3D 11:** This value is not supported until Direct3D 11.1.

## Remarks

In general, binding flags can be combined using a bitwise OR (except the constant-buffer flag); however, you should use a single flag to allow the device to optimize the resource usage.

This enumeration is used by a:

- [Buffer description](#) when creating a buffer.
- Texture description when creating a texture (see [D3D11\\_TEXTURE1D\\_DESC](#) or [D3D11\\_TEXTURE2D\\_DESC](#) or [D3D11\\_TEXTURE3D\\_DESC](#)).

A shader-resource buffer is NOT a constant buffer; rather, it is a texture or buffer resource that is bound to a shader, that contains texture or buffer data (it is not limited to a single element type in the buffer). A shader-resource buffer is created with the D3D11\_BIND\_SHADER\_RESOURCE flag and is bound to the pipeline using one of these APIs: [ID3D11DeviceContext::GSSetShaderResources](#), [ID3D11DeviceContext::PSSetShaderResources](#), or [ID3D11DeviceContext::VSSetShaderResources](#). Furthermore, a shader-resource buffer cannot use the [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#) flag.

**Note** The Direct3D 11.1 runtime, which is available starting with Windows 8, enables mapping dynamic constant buffers and shader resource views (SRVs) of dynamic buffers with [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#). The Direct3D 11 and earlier runtimes limited mapping to vertex or index buffers. To determine if a Direct3D device supports these features, call [ID3D11Device::CheckFeatureSupport](#) with [D3D11\\_FEATURE\\_D3D11\\_OPTIONS](#). [CheckFeatureSupport](#) fills members of a [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) structure with the device's features. The relevant members here are [MapNoOverwriteOnDynamicConstantBuffer](#) and [MapNoOverwriteOnDynamicBufferSRV](#).

# Requirements

Header	d3d11.h
--------	---------

## See also

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BUFFEREX\_SRV\_FLAG enumeration (d3d11.h)

Article 02/22/2024

Identifies how to view a buffer resource.

## Syntax

C++

```
typedef enum D3D11_BUFFEREX_SRV_FLAG {
    D3D11_BUFFEREX_SRV_FLAG_RAW = 0x1
} ;
```

## Constants

[+] Expand table

D3D11\_BUFFEREX\_SRV\_FLAG\_RAW

Value: 0x1

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

## Remarks

This enumeration is used by [D3D11\\_BUFFEREX\\_SRV](#)

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_BUFFER\_UAV\_FLAG enumeration (d3d11.h)

Article02/22/2024

Identifies unordered-access view options for a buffer resource.

## Syntax

C++

```
typedef enum D3D11_BUFFER_UAV_FLAG {
    D3D11_BUFFER_UAV_FLAG_RAW = 0x1,
    D3D11_BUFFER_UAV_FLAG_APPEND = 0x2,
    D3D11_BUFFER_UAV_FLAG_COUNTER = 0x4
} ;
```

## Constants

[+] Expand table

`D3D11_BUFFER_UAV_FLAG_RAW`

Value: `0x1`

Resource contains raw, unstructured data. Requires the UAV format to be `DXGI_FORMAT_R32_TYPELESS`.

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

`D3D11_BUFFER_UAV_FLAG_APPEND`

Value: `0x2`

Allow data to be appended to the end of the buffer. `D3D11_BUFFER_UAV_FLAG_APPEND` flag must also be used for

any view that will be used as a [AppendStructuredBuffer](#) or a [ConsumeStructuredBuffer](#).

Requires the UAV format to be `DXGI_FORMAT_UNKNOWN`.

`D3D11_BUFFER_UAV_FLAG_COUNTER`

Value: `0x4`

Adds a counter to the unordered-access-view buffer. `D3D11_BUFFER_UAV_FLAG_COUNTER` can only be used on a UAV that is a

[RWStructuredBuffer](#) and it enables the functionality needed for the [IncrementCounter](#) and [DecrementCounter](#) methods in HLSL. Requires the UAV format to be `DXGI_FORMAT_UNKNOWN`.

# Requirements

 Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_CHECK\_MULTISAMPLE\_QUALITY\_LEVELS\_FLAG enumeration (d3d11\_2.h)

Article02/22/2024

Identifies how to check multisample quality levels.

## Syntax

C++

```
typedef enum D3D11_CHECK_MULTISAMPLE_QUALITY_LEVELS_FLAG {
    D3D11_CHECK_MULTISAMPLE_QUALITY_LEVELS_TILED_RESOURCE = 0x1
} ;
```

## Constants

[+] Expand table

D3D11\_CHECK\_MULTISAMPLE\_QUALITY\_LEVELS\_TILED\_RESOURCE

Value: 0x1

Indicates to check the multisample quality levels of a tiled resource.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11_2.h

## See also

[Resource Enumerations](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_CPU\_ACCESS\_FLAG enumeration (d3d11.h)

Article 02/22/2024

Specifies the types of CPU access allowed for a resource.

## Syntax

C++

```
typedef enum D3D11_CPU_ACCESS_FLAG {
    D3D11_CPU_ACCESS_WRITE = 0x10000L,
    D3D11_CPU_ACCESS_READ = 0x20000L
};
```

## Constants

[] Expand table

<code>D3D11_CPU_ACCESS_WRITE</code>
-------------------------------------

Value: *0x10000L*

The resource is to be mappable so that the CPU can change its contents. Resources created with this flag cannot be set as outputs of the pipeline and must be created with either dynamic or staging usage (see [D3D11\\_USAGE](#)).

<code>D3D11_CPU_ACCESS_READ</code>
------------------------------------

Value: *0x20000L*

The resource is to be mappable so that the CPU can read its contents. Resources created with this flag cannot be set as either inputs or outputs to the pipeline and must be created with staging usage (see [D3D11\\_USAGE](#)).

## Remarks

This enumeration is used in [D3D11\\_BUFFER\\_DESC](#), [D3D11\\_TEXTURE1D\\_DESC](#), [D3D11\\_TEXTURE2D\\_DESC](#), [D3D11\\_TEXTURE3D\\_DESC](#).

Applications may combine one or more of these flags with a bitwise OR. When possible, create resources with no CPU access flags, as this enables better resource optimization.

The D3D11\_RESOURCE\_MISC\_FLAG cannot be used when creating resources with D3D11\_CPU\_ACCESS flags.

## Requirements

[Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_DSV\_DIMENSION enumeration (d3d11.h)

Article 02/22/2024

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

## Syntax

C++

```
typedef enum D3D11_DSV_DIMENSION {
    D3D11_DSV_DIMENSION_UNKNOWN = 0,
    D3D11_DSV_DIMENSION_TEXTURE1D = 1,
    D3D11_DSV_DIMENSION_TEXTURE1DARRAY = 2,
    D3D11_DSV_DIMENSION_TEXTURE2D = 3,
    D3D11_DSV_DIMENSION_TEXTURE2DARRAY = 4,
    D3D11_DSV_DIMENSION_TEXTURE2DMS = 5,
    D3D11_DSV_DIMENSION_TEXTURE2DMSARRAY = 6
};
```

## Constants

[+] Expand table

`D3D11_DSV_DIMENSION_UNKNOWN`

Value: 0

`D3D11_DSV_DIMENSION_UNKNOWN` is not a valid value for `D3D11_DEPTH_STENCIL_VIEW_DESC` and is not used.

`D3D11_DSV_DIMENSION_TEXTURE1D`

Value: 1

The resource will be accessed as a 1D texture.

`D3D11_DSV_DIMENSION_TEXTURE1DARRAY`

Value: 2

The resource will be accessed as an array of 1D textures.

`D3D11_DSV_DIMENSION_TEXTURE2D`

Value: 3

The resource will be accessed as a 2D texture.

**D3D11\_DSV\_DIMENSION\_TEXTURE2DARRAY**

Value: 4

The resource will be accessed as an array of 2D textures.

**D3D11\_DSV\_DIMENSION\_TEXTURE2DMS**

Value: 5

The resource will be accessed as a 2D texture with multisampling.

**D3D11\_DSV\_DIMENSION\_TEXTURE2DMSARRAY**

Value: 6

The resource will be accessed as an array of 2D textures with multisampling.

## Remarks

This enumeration is used in [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) to create a depth-stencil view.

## Requirements

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

Requirement	Value
Header	d3d11.h

## See also

[Resource Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_DSV\_FLAG enumeration (d3d11.h)

Article 02/22/2024

Depth-stencil view options.

## Syntax

C++

```
typedef enum D3D11_DSV_FLAG {
    D3D11_DSV_READ_ONLY_DEPTH = 0x1L,
    D3D11_DSV_READ_ONLY_STENCIL = 0x2L
};
```

## Constants

[] [Expand table](#)

`D3D11_DSV_READ_ONLY_DEPTH`

Value: `0x1L`

Indicates that depth values are read only.

`D3D11_DSV_READ_ONLY_STENCIL`

Value: `0x2L`

Indicates that stencil values are read only.

## Remarks

This enumeration is used by [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#).

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 a read/write conflicts between separate views.

## Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_MAP enumeration (d3d11.h)

Article 07/27/2022

Identifies a resource to be accessed for reading and writing by the CPU. Applications may combine one or more of these flags.

## Syntax

C++

```
typedef enum D3D11_MAP {
    D3D11_MAP_READ = 1,
    D3D11_MAP_WRITE = 2,
    D3D11_MAP_READ_WRITE = 3,
    D3D11_MAP_WRITE_DISCARD = 4,
    D3D11_MAP_WRITE_NO_OVERWRITE = 5
};
```

## Constants

`D3D11_MAP_READ`

Value: 1

Resource is mapped for reading. The resource must have been created with read access (see [D3D11\\_CPU\\_ACCESS\\_READ](#)).

`D3D11_MAP_WRITE`

Value: 2

Resource is mapped for writing. The resource must have been created with write access (see [D3D11\\_CPU\\_ACCESS\\_WRITE](#)).

`D3D11_MAP_READ_WRITE`

Value: 3

Resource is mapped for reading and writing. The resource must have been created with read and write access (see [D3D11\\_CPU\\_ACCESS\\_READ](#) and [D3D11\\_CPU\\_ACCESS\\_WRITE](#)).

`D3D11_MAP_WRITE_DISCARD`

Value: 4

Resource is mapped for writing; the previous contents of the resource will be undefined. The resource must have been created with write access and dynamic usage (See [D3D11\\_CPU\\_ACCESS\\_WRITE](#) and [D3D11\\_USAGE\\_DYNAMIC](#)).

#### D3D11\_MAP\_WRITE\_NO\_OVERWRITE

Value: 5

Resource is mapped for writing; the existing contents of the resource cannot be overwritten (see Remarks). This flag is only valid on vertex and index buffers. The resource must have been created with write access (see [D3D11\\_CPU\\_ACCESS\\_WRITE](#)).

Cannot be used on a resource created with the [D3D11\\_BIND\\_CONSTANT\\_BUFFER](#) flag.

**Note** The Direct3D 11.1 runtime, which is available starting with Windows 8, enables mapping dynamic constant buffers and shader resource views (SRVs) of dynamic buffers with **D3D11\_MAP\_WRITE\_NO\_OVERWRITE**. The Direct3D 11 and earlier runtimes limited mapping to vertex or index buffers. To determine if a Direct3D device supports these features, call [ID3D11Device::CheckFeatureSupport](#) with [D3D11\\_FEATURE\\_D3D11\\_OPTIONS](#). [CheckFeatureSupport](#) fills members of a [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS](#) structure with the device's features. The relevant members here are [MapNoOverwriteOnDynamicConstantBuffer](#) and [MapNoOverwriteOnDynamicBufferSRV](#).

## Remarks

This enumeration is used in [ID3D11DeviceContext::Map](#).

## Meaning of D3D11\_MAP\_WRITE\_NO\_OVERWRITE

**D3D11\_MAP\_WRITE\_NO\_OVERWRITE** signifies that the application promises not to write to data that the input assembler (IA) stage is using. In exchange, the GPU allows the application to write to other parts of the same buffer. The application must ensure that it does not write over any data in use by the IA stage.

For example, consider the buffer illustrated in the following diagram. If a [Draw](#) call has been issued that uses vertices 4-6, then an application that calls [Map](#) on this buffer must ensure that it does not write to the vertices that the [Draw](#) call will access during rendering.

Initialized							In Use						Uninitialized				
V, N U, V																	
0	1	2	3	4	5	6	7	8	9	10	11	12					

However, ensuring this can be difficult, because the GPU is often many frames behind the CPU in terms of which frame it is currently processing. Keeping track of which sections of a resource are being used because of calls made 2 to 5 frames ago is difficult and error-prone. Because of this, it is recommended that applications only write to the uninitialized portions of a resource when using `D3D11_MAP_WRITE_NO_OVERWRITE`.

## Common Usage of `D3D11_MAP_WRITE_DISCARD` with `D3D11_MAP_WRITE_NO_OVERWRITE`

`D3D11_MAP_WRITE_DISCARD` and `D3D11_MAP_WRITE_NO_OVERWRITE` are normally used in conjunction with dynamic index/vertex buffers. `D3D11_MAP_WRITE_DISCARD` can also be used with dynamic textures. However, `D3D11_MAP_WRITE_NO_OVERWRITE` cannot be used with dynamic textures.

A common use of these two flags involves filling dynamic index/vertex buffers with geometry that can be seen from the camera's current position. The first time that data is entered into the buffer on a given frame, [Map](#) is called with `D3D11_MAP_WRITE_DISCARD`; doing so invalidates the previous contents of the buffer. The buffer is then filled with all available data.

Subsequent writes to the buffer within the same frame should use `D3D11_MAP_WRITE_NO_OVERWRITE`. This will enable the CPU to access a resource that is potentially being used by the GPU as long as the restrictions described previously are respected.

## Requirements

Header	d3d11.h
--------	---------

## See also

[Resource Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_MAP\_FLAG enumeration (d3d11.h)

Article02/22/2024

Specifies how the CPU should respond when an application calls the [ID3D11DeviceContext::Map](#) method on a resource that is being used by the GPU.

## Syntax

C++

```
typedef enum D3D11_MAP_FLAG {
    D3D11_MAP_FLAG_DO_NOT_WAIT = 0x100000L
} ;
```

## Constants

[+] [Expand table](#)

D3D11\_MAP\_FLAG\_DO\_NOT\_WAIT

Value: *0x100000L*

Specifies that [ID3D11DeviceContext::Map](#) should return [DXGI\\_ERROR\\_WAS\\_STILL\\_DRAWING](#) when the GPU blocks the CPU from accessing a resource. For more information about this error code, see [DXGI\\_ERROR](#).

## Remarks

This enumeration is used by [ID3D11DeviceContext::Map](#).

D3D11\_MAP\_FLAG\_DO\_NOT\_WAIT cannot be used with [D3D11\\_MAP\\_WRITE\\_DISCARD](#) or [D3D11\\_MAP\\_WRITE\\_NOOVERWRITE](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_RESOURCE\_DIMENSION enumeration (d3d11.h)

Article 02/22/2024

Identifies the type of resource being used.

## Syntax

C++

```
typedef enum D3D11_RESOURCE_DIMENSION {
    D3D11_RESOURCE_DIMENSION_UNKNOWN = 0,
    D3D11_RESOURCE_DIMENSION_BUFFER = 1,
    D3D11_RESOURCE_DIMENSION_TEXTURE1D = 2,
    D3D11_RESOURCE_DIMENSION_TEXTURE2D = 3,
    D3D11_RESOURCE_DIMENSION_TEXTURE3D = 4
};
```

## Constants

[] Expand table

	<code>D3D11_RESOURCE_DIMENSION_UNKNOWN</code>
Value:	0
Resource is of unknown type.	
	<code>D3D11_RESOURCE_DIMENSION_BUFFER</code>
Value:	1
Resource is a buffer.	
	<code>D3D11_RESOURCE_DIMENSION_TEXTURE1D</code>
Value:	2
Resource is a 1D texture.	
	<code>D3D11_RESOURCE_DIMENSION_TEXTURE2D</code>
Value:	3
Resource is a 2D texture.	
	<code>D3D11_RESOURCE_DIMENSION_TEXTURE3D</code>
Value:	4
Resource is a 3D texture.	

## Remarks

This enumeration is used in [ID3D11Resource::GetType](#).

## Requirements

 [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_RESOURCE\_MISC\_FLAG enumeration (d3d11.h)

Article07/06/2022

Identifies options for resources.

## Syntax

C++

```
typedef enum D3D11_RESOURCE_MISC_FLAG {
    D3D11_RESOURCE_MISC_GENERATE_MIPS = 0x1L,
    D3D11_RESOURCE_MISC_SHARED = 0x2L,
    D3D11_RESOURCE_MISC_TEXTURECUBE = 0x4L,
    D3D11_RESOURCE_MISC_DRAWINDIRECT_ARGS = 0x10L,
    D3D11_RESOURCE_MISC_BUFFER_ALLOW_RAW_VIEWS = 0x20L,
    D3D11_RESOURCE_MISC_BUFFER_STRUCTURED = 0x40L,
    D3D11_RESOURCE_MISC_RESOURCE_CLAMP = 0x80L,
    D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX = 0x100L,
    D3D11_RESOURCE_MISC_GDI_COMPATIBLE = 0x200L,
    D3D11_RESOURCE_MISC_SHARED_NTHANDLE = 0x800L,
    D3D11_RESOURCE_MISC_RESTRICTED_CONTENT = 0x1000L,
    D3D11_RESOURCE_MISC_RESTRICT_SHARED_RESOURCE = 0x2000L,
    D3D11_RESOURCE_MISC_RESTRICT_SHARED_RESOURCE_DRIVER = 0x4000L,
    D3D11_RESOURCE_MISC_GUARDED = 0x8000L,
    D3D11_RESOURCE_MISC_TILE_POOL = 0x20000L,
    D3D11_RESOURCE_MISC_TILED = 0x40000L,
    D3D11_RESOURCE_MISC_HW_PROTECTED = 0x80000L,
    D3D11_RESOURCE_MISC_SHARED_DISPLAYABLE,
    D3D11_RESOURCE_MISC_SHARED_EXCLUSIVE_WRITER
} ;
```

## Constants

D3D11\_RESOURCE\_MISC\_GENERATE\_MIPS

Value: *0x1L*

Enables MIP map generation by using [ID3D11DeviceContext::GenerateMips](#) on a texture resource. The resource must be created with the [bind flags](#) that specify that the resource is a render target and a shader resource.

**D3D11\_RESOURCE\_MISC\_SHARED**

Value: *0x2L*

Enables resource data sharing between two or more Direct3D devices. The only resources that can be shared are 2D non-mipmapped textures.

**D3D11\_RESOURCE\_MISC\_SHARED** and **D3D11\_RESOURCE\_MISC\_SHARED\_KEYEDMUTEX** are mutually exclusive.

**WARP** and **REF** devices do not support shared resources.

If you try to create a resource with this flag on either a **WARP** or **REF** device, the create method will return an **E\_OUTOFMEMORY** error code.

**Note** Starting with Windows 8, **WARP** devices fully support shared resources.

**Note** Starting with Windows 8, we recommend that you enable resource data sharing between two or more Direct3D devices by using a combination of the **D3D11\_RESOURCE\_MISC\_SHARED\_NTHANDLE** and **D3D11\_RESOURCE\_MISC\_SHARED\_KEYEDMUTEX** flags instead.

**D3D11\_RESOURCE\_MISC\_TEXTURECUBE**

Value: *0x4L*

Sets a resource to be a cube texture created from a [Texture2DArray](#) that contains 6 textures.

**D3D11\_RESOURCE\_MISC\_DRAWINDIRECT\_ARGS**

Value: *0x10L*

Enables instancing of GPU-generated content.

**D3D11\_RESOURCE\_MISC\_BUFFER\_ALLOW\_RAW\_VIEWS**

Value: *0x20L*

Enables a resource as a [byte address buffer](#).

**D3D11\_RESOURCE\_MISC\_BUFFER\_STRUCTURED**

Value: *0x40L*

Enables a resource as a [structured buffer](#).

`D3D11_RESOURCE_MISC_RESOURCE_CLAMP`

Value: `0x80L`

Enables a resource with MIP map clamping for use with [ID3D11DeviceContext::SetResourceMinLOD](#).

`D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX`

Value: `0x100L`

Enables the resource to be synchronized by using the [IDXGIFKeyedMutex::AcquireSync](#) and [IDXGIFKeyedMutex::ReleaseSync](#) APIs.

The following Direct3D 11 resource creation APIs, that take `D3D11_RESOURCE_MISC_FLAG` parameters, have been extended to support the new flag.

- [ID3D11Device::CreateTexture1D](#)
- [ID3D11Device::CreateTexture2D](#)
- [ID3D11Device::CreateTexture3D](#)
- [ID3D11Device::CreateBuffer](#)

If you call any of these methods with the `D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX` flag set, the interface returned will support the [IDXGIFKeyedMutex](#) interface. You can retrieve a pointer to the [IDXGIFKeyedMutex](#) interface from the resource by using [IUnknown::QueryInterface](#). The [IDXGIFKeyedMutex](#) interface implements the [IDXGIFKeyedMutex::AcquireSync](#) and [IDXGIFKeyedMutex::ReleaseSync](#) APIs to synchronize access to the surface. The device that creates the surface, and any other device that opens the surface by using [OpenSharedResource](#), must call [IDXGIFKeyedMutex::AcquireSync](#) before they issue any rendering commands to the surface. When those devices finish rendering, they must call [IDXGIFKeyedMutex::ReleaseSync](#).

`D3D11_RESOURCE_MISC_SHARED` and `D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX` are mutually exclusive.

**WARP** and **REF** devices do not support shared resources.

If you try to create a resource with this flag on either a **WARP** or **REF** device, the create method will return an `E_OUTOFMEMORY` error code.

**Note** Starting with Windows 8, WARP devices fully support shared resources.

**D3D11\_RESOURCE\_MISC\_GDI\_COMPATIBLE**

Value: 0x200L

Enables a resource compatible with GDI. You must set the **D3D11\_RESOURCE\_MISC\_GDI\_COMPATIBLE** flag on surfaces that you use with GDI. Setting the **D3D11\_RESOURCE\_MISC\_GDI\_COMPATIBLE** flag allows GDI rendering on the surface via [IDXGISurface1::GetDC](#).

Consider the following programming tips for using **D3D11\_RESOURCE\_MISC\_GDI\_COMPATIBLE** when you create a texture or use that texture in a swap chain:

- **D3D11\_RESOURCE\_MISC\_SHARED\_KEYEDMUTEX** and **D3D11\_RESOURCE\_MISC\_GDI\_COMPATIBLE** are mutually exclusive. Therefore, do not use them together.
- **D3D11\_RESOURCE\_MISC\_RESOURCE\_CLAMP** and **D3D11\_RESOURCE\_MISC\_GDI\_COMPATIBLE** are mutually exclusive. Therefore, do not use them together.
- You must bind the texture as a render target for the output-merger stage. For example, set the **D3D11\_BIND\_RENDER\_TARGET** flag in the **BindFlags** member of the [D3D11\\_TEXTURE2D\\_DESC](#) structure.
- You must set the maximum number of MIP map levels to 1. For example, set the **MipLevels** member of the [D3D11\\_TEXTURE2D\\_DESC](#) structure to 1.
- You must specify that the texture requires read and write access by the GPU. For example, set the **Usage** member of the [D3D11\\_TEXTURE2D\\_DESC](#) structure to **D3D11\_USAGE\_DEFAULT**.
  - You must set the texture format to one of the following types.
    - **DXGI\_FORMAT\_B8G8R8A8\_UNORM**
    - **DXGI\_FORMAT\_B8G8R8A8\_TYPELESS**
    - **DXGI\_FORMAT\_B8G8R8A8\_UNORM\_SRGB**

For example, set the **Format** member of the [D3D11\\_TEXTURE2D\\_DESC](#) structure to one of these types.

- You cannot use D3D11\_RESOURCE\_MISC\_GDI\_COMPATIBLE with multisampling. Therefore, set the **Count** member of the [DXGI\\_SAMPLE\\_DESC](#) structure to 1. Then, set the **SampleDesc** member of the [D3D11\\_TEXTURE2D\\_DESC](#) structure to this [DXGI\\_SAMPLE\\_DESC](#) structure.

#### D3D11\_RESOURCE\_MISC\_SHARED\_NTHANDLE

Value: *0x800L*

Set this flag to enable the use of NT HANDLE values when you create a shared resource. By enabling this flag, you deprecate the use of existing HANDLE values.

The value specifies a new shared resource type that directs the runtime to use NT HANDLE values for the shared resource. The runtime then must confirm that the shared resource works on all hardware at the specified [feature level](#).

Without this flag set, the runtime does not strictly validate shared resource parameters (that is, formats, flags, usage, and so on). When the runtime does not validate shared resource parameters, behavior of much of the Direct3D API might be undefined and might vary from driver to driver.

**Direct3D 11 and earlier:** This value is not supported until Direct3D 11.1.

#### D3D11\_RESOURCE\_MISC\_RESTRICTED\_CONTENT

Value: *0x1000L*

Set this flag to indicate that the resource might contain protected content; therefore, the operating system should use the resource only when the driver and hardware support content protection. If the driver and hardware do not support content protection and you try to create a resource with this flag, the resource creation fails.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

#### D3D11\_RESOURCE\_MISC\_RESTRICT\_SHARED\_RESOURCE

Value: *0x2000L*

Set this flag to indicate that the operating system restricts access to the shared surface. You can use this flag together with the D3D11\_RESOURCE\_MISC\_RESTRICT\_SHARED\_RESOURCE\_DRIVER flag and only when you create a shared surface. The process that creates the shared resource can always open the shared resource.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_RESOURCE\_MISC\_RESTRICT\_SHARED\_RESOURCE\_DRIVER**

Value: *0x4000L*

Set this flag to indicate that the driver restricts access to the shared surface. You can use this flag in conjunction with the D3D11\_RESOURCE\_MISC\_RESTRICT\_SHARED\_RESOURCE flag and only when you create a shared surface. The process that creates the shared resource can always open the shared resource.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_RESOURCE\_MISC\_GUARDED**

Value: *0x8000L*

Set this flag to indicate that the resource is guarded. Such a resource is returned by the [IDCompositionSurface::BeginDraw](#) (DirectComposition) and [ISurfaceImageSourceNative::BeginDraw](#) (Windows Runtime) APIs. For these APIs, you provide a region of interest (ROI) on a surface to update. This surface isn't compatible with multiple render targets (MRT).

A guarded resource automatically restricts all writes to the region that is related to one of the preceding APIs. Additionally, the resource enforces access to the ROI with these restrictions:

- Copy operations from the resource by using [ID3D11DeviceContext::CopyResource](#) or [ID3D11DeviceContext::CopySubresourceRegion](#) are restricted to only copy from the ROI.
- When a guarded resource is set as a render target, it must be the only target.

**Direct3D 11:** This value is not supported until Direct3D 11.1.

**D3D11\_RESOURCE\_MISC\_TILE\_POOL**

Value: *0x20000L*

Set this flag to indicate that the resource is a tile pool.

**Direct3D 11:** This value is not supported until Direct3D 11.2.

**D3D11\_RESOURCE\_MISC\_TILED**

Value: *0x40000L*

Set this flag to indicate that the resource is a tiled resource.

**Direct3D 11:** This value is not supported until Direct3D 11.2.

`D3D11_RESOURCE_MISC_HW_PROTECTED`

Value: `0x80000L`

Set this flag to indicate that the resource should be created such that it will be protected by the hardware. Resource creation will fail if hardware content protection is not supported.

This flag has the following restrictions:

- This flag cannot be used with the following [D3D11\\_USAGE](#) values:
  - `D3D11_USAGE_DYNAMIC`
  - `D3D11_USAGE_STAGING`
- This flag cannot be used with the following [D3D11\\_BIND\\_FLAG](#) values.
  - `D3D11_BIND_VERTEX_BUFFER`
  - `D3D11_BIND_INDEX_BUFFER`
- No CPU access flags can be specified.

**Note**

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

**Note** This enumeration value is supported starting with Windows 10.

#### D3D11\_RESOURCE\_MISC\_SHARED\_DISPLAYABLE

Enables the resource to work with the [displayable surfaces](#) feature. You must use

**D3D11\_RESOURCE\_MISC\_SHARED\_DISPLAYABLE** in combination with both

**D3D11\_RESOURCE\_MISC\_SHARED** and **D3D11\_RESOURCE\_MISC\_SHARED\_NTHANDLE**.

#### D3D11\_RESOURCE\_MISC\_SHARED\_EXCLUSIVE\_WRITER

TBD

## Remarks

This enumeration is used in [D3D11\\_BUFFER\\_DESC](#), [D3D11\\_TEXTURE1D\\_DESC](#), [D3D11\\_TEXTURE2D\\_DESC](#), [D3D11\\_TEXTURE3D\\_DESC](#).

These flags can be combined by bitwise OR.

The **D3D11\_RESOURCE\_MISC\_FLAG** cannot be used when creating resources with **D3D11\_CPU\_ACCESS** flags.

## Requirements

Header

d3d11.h

## See also

[Resource Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_RTV\_DIMENSION enumeration (d3d11.h)

Article 01/31/2022

These flags identify the type of resource that will be viewed as a render target.

## Syntax

C++

```
typedef enum D3D11_RTV_DIMENSION {
    D3D11_RTV_DIMENSION_UNKNOWN = 0,
    D3D11_RTV_DIMENSION_BUFFER = 1,
    D3D11_RTV_DIMENSION_TEXTURE1D = 2,
    D3D11_RTV_DIMENSION_TEXTURE1DARRAY = 3,
    D3D11_RTV_DIMENSION_TEXTURE2D = 4,
    D3D11_RTV_DIMENSION_TEXTURE2DARRAY = 5,
    D3D11_RTV_DIMENSION_TEXTURE2DMS = 6,
    D3D11_RTV_DIMENSION_TEXTURE2DMSARRAY = 7,
    D3D11_RTV_DIMENSION_TEXTURE3D = 8
} ;
```

## Constants

D3D11\_RTV\_DIMENSION\_UNKNOWN

Value: 0

Do not use this value, as it will cause [ID3D11Device::CreateRenderTargetView](#) to fail.

D3D11\_RTV\_DIMENSION\_BUFFER

Value: 1

The resource will be accessed as a buffer.

D3D11\_RTV\_DIMENSION\_TEXTURE1D

Value: 2

The resource will be accessed as a 1D texture.

D3D11\_RTV\_DIMENSION\_TEXTURE1DARRAY

Value: 3

The resource will be accessed as an array of 1D textures.

`D3D11_RTVE_DIMENSION_TEXTURE2D`

Value: 4

The resource will be accessed as a 2D texture.

`D3D11_RTVE_DIMENSION_TEXTURE2DARRAY`

Value: 5

The resource will be accessed as an array of 2D textures.

`D3D11_RTVE_DIMENSION_TEXTURE2DMS`

Value: 6

The resource will be accessed as a 2D texture with multisampling.

`D3D11_RTVE_DIMENSION_TEXTURE2DMSARRAY`

Value: 7

The resource will be accessed as an array of 2D textures with multisampling.

`D3D11_RTVE_DIMENSION_TEXTURE3D`

Value: 8

The resource will be accessed as a 3D texture.

## Remarks

This enumeration is used in [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) to create a render-target view.

## Requirements

Header

d3d11.h

## See also

[Resource Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_SRV\_DIMENSION enumeration

Article 12/01/2017

These flags identify the type of resource that will be viewed as a shader resource.

## Syntax

```
c++  
  
typedef enum D3D11_SRV_DIMENSION {  
    D3D11_SRV_DIMENSION_UNKNOWN          = 0,  
    D3D11_SRV_DIMENSION_BUFFER           = 1,  
    D3D11_SRV_DIMENSION_TEXTURE1D        = 2,  
    D3D11_SRV_DIMENSION_TEXTURE1DARRAY   = 3,  
    D3D11_SRV_DIMENSION_TEXTURE2D        = 4,  
    D3D11_SRV_DIMENSION_TEXTURE2DARRAY   = 5,  
    D3D11_SRV_DIMENSION_TEXTURE2DMS      = 6,  
    D3D11_SRV_DIMENSION_TEXTURE2DMSARRAY = 7,  
    D3D11_SRV_DIMENSION_TEXTURE3D        = 8,  
    D3D11_SRV_DIMENSION_TEXTURECUBE      = 9,  
    D3D11_SRV_DIMENSION_TEXTURECUBEARRAY = 10,  
    D3D11_SRV_DIMENSION_BUFFEREX         = 11  
} D3D11_SRV_DIMENSION;
```

## Constants

- **D3D11\_SRV\_DIMENSION\_UNKNOWN**  
The type is unknown.
- **D3D11\_SRV\_DIMENSION\_BUFFER**  
The resource is a buffer.
- **D3D11\_SRV\_DIMENSION\_TEXTURE1D**  
The resource is a 1D texture.
- **D3D11\_SRV\_DIMENSION\_TEXTURE1DARRAY**  
The resource is an array of 1D textures.
- **D3D11\_SRV\_DIMENSION\_TEXTURE2D**  
The resource is a 2D texture.
- **D3D11\_SRV\_DIMENSION\_TEXTURE2DARRAY**  
The resource is an array of 2D textures.

- **D3D11\_SRV\_DIMENSION\_TEXTURE2DMS**  
The resource is a multisampling 2D texture.
- **D3D11\_SRV\_DIMENSION\_TEXTURE2DMSARRAY**  
The resource is an array of multisampling 2D textures.
- **D3D11\_SRV\_DIMENSION\_TEXTURE3D**  
The resource is a 3D texture.
- **D3D11\_SRV\_DIMENSION\_TEXTURECUBE**  
The resource is a cube texture.
- **D3D11\_SRV\_DIMENSION\_TEXTURECUBEARRAY**  
The resource is an array of cube textures.
- **D3D11\_SRV\_DIMENSION\_BUFFEREX**  
The resource is a raw buffer. For more info about raw viewing of buffers, see [Raw Views of Buffers](#).

## Remarks

These flags are used by a shader-resource-view description (see [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)).

The **D3D11\_SRV\_DIMENSION** enumeration is type defined in the D3D11.h header file as a [D3D\\_SRV\\_DIMENSION](#) enumeration, which is fully defined in the D3DCCommon.h header file.

```
typedef D3D_SRV_DIMENSION D3D11_SRV_DIMENSION;
```

## Requirements

 Expand table

Header	D3D11.h
--------	---------

## See also

[Resource Enumerations](#)

# D3D11\_STANDARD\_MULTISAMPLE\_QUALITY\_LEVELS enumeration (d3d11.h)

Article 02/22/2024

Specifies a multi-sample pattern type.

## Syntax

C++

```
typedef enum D3D11_STANDARD_MULTISAMPLE_QUALITY_LEVELS {
    D3D11_STANDARD_MULTISAMPLE_PATTERN = 0xffffffff,
    D3D11_CENTER_MULTISAMPLE_PATTERN = 0xfffffff
} ;
```

## Constants

[+] Expand table

**D3D11\_STANDARD\_MULTISAMPLE\_PATTERN**

Value: *0xffffffff*

Pre-defined multi-sample patterns required for Direct3D 11 and Direct3D 10.1 hardware.

**D3D11\_CENTER\_MULTISAMPLE\_PATTERN**

Value: *0xfffffff*

Pattern where all of the samples are located at the pixel center.

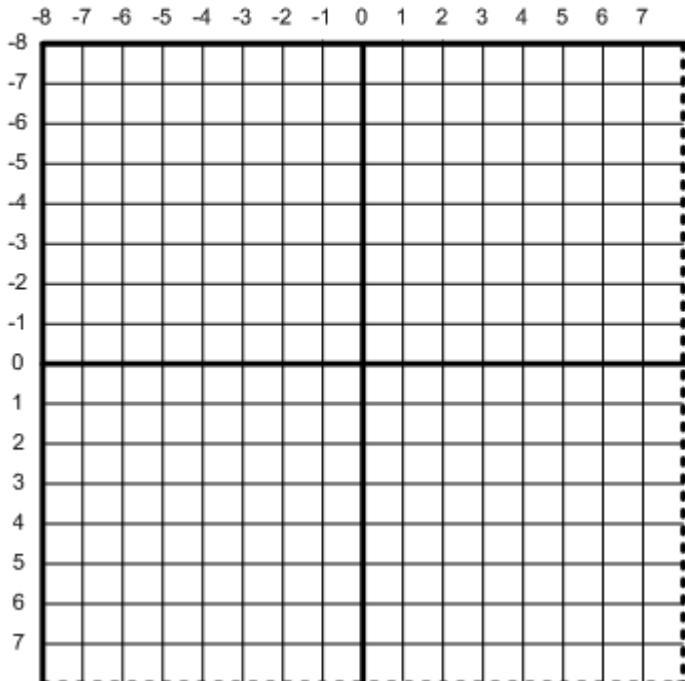
## Remarks

An app calls [ID3D11Device::CheckMultisampleQualityLevels](#) to get the number of quality levels available during multisampling. A 0 quality level means the hardware does not support multisampling for the particular format. If the number of quality levels is greater than 0 and the hardware supports the fixed sample patterns for the sample count, the app can request the fixed patterns by specifying quality level as either **D3D11\_STANDARD\_MULTISAMPLE\_PATTERN** or **D3D11\_CENTER\_MULTISAMPLE\_PATTERN**. The app can call [ID3D11Device::CheckFormatSupport](#) to check for support of the standard fixed patterns. If the hardware only supports the fixed patterns but no additional vendor-specific

patterns, the runtime can report the number of quality levels as 1, and the hardware can pretend 0 quality level behaves the same as quality level equal to D3D11\_STANDARD\_MULTISAMPLE\_PATTERN.

The runtime defines the following standard sample patterns for 1(trivial), 2, 4, 8, and 16 sample counts. Hardware must support 1, 4, and 8 sample counts. Hardware vendors can expose more sample counts beyond these. However, if vendors support 2, 4(required), 8(required), or 16, they must also support the corresponding standard pattern or center pattern for each of those sample counts.

## Sample Coordinate System



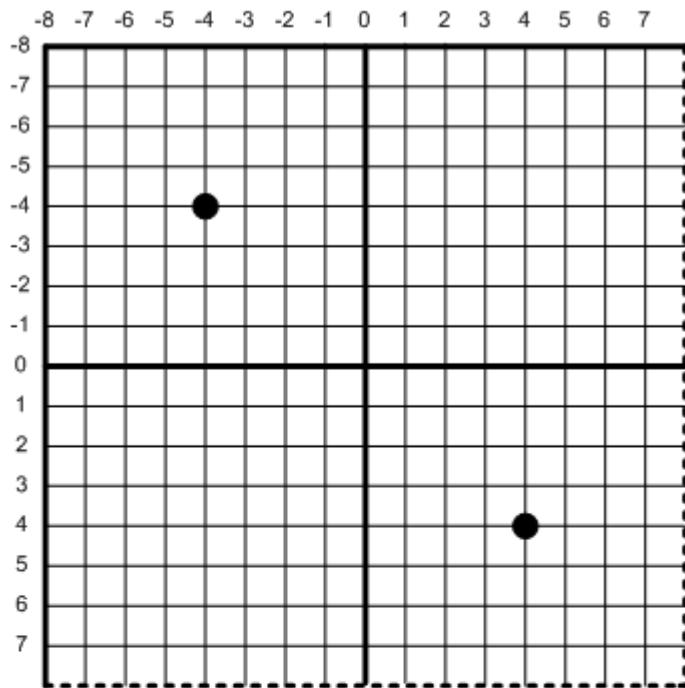
Shown is a pixel's area, center at (0,0).

Only coordinates [-8,7] are valid for both axis.

Coordinates are in 1/16ths;  
discrete sample (i,j) has  
continuous coordinates (i/16,j/16)

---

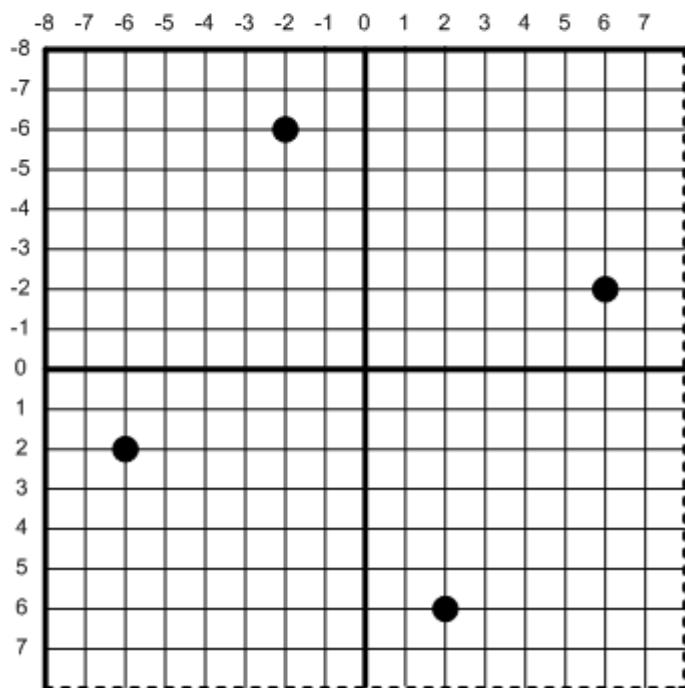
## Standard 2 Sample Pattern



● { (4,4),(-4,-4) }

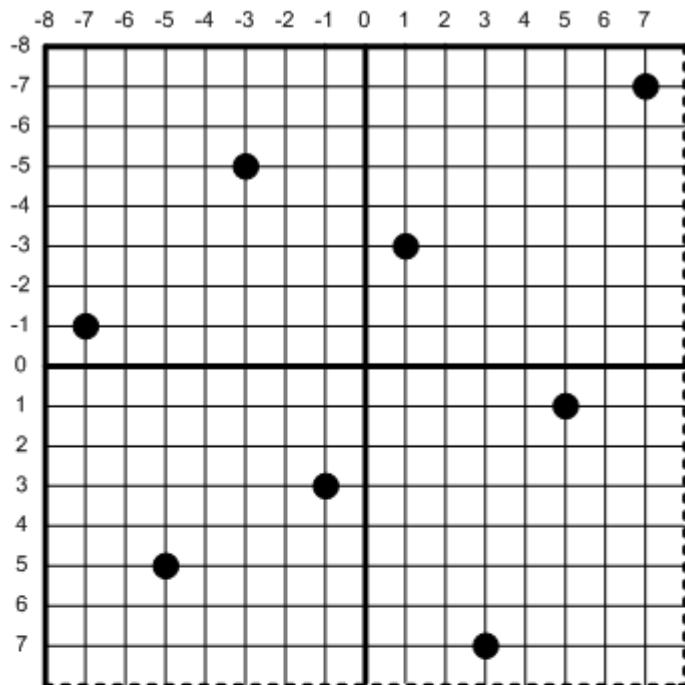
---

## Standard 4 Sample Pattern



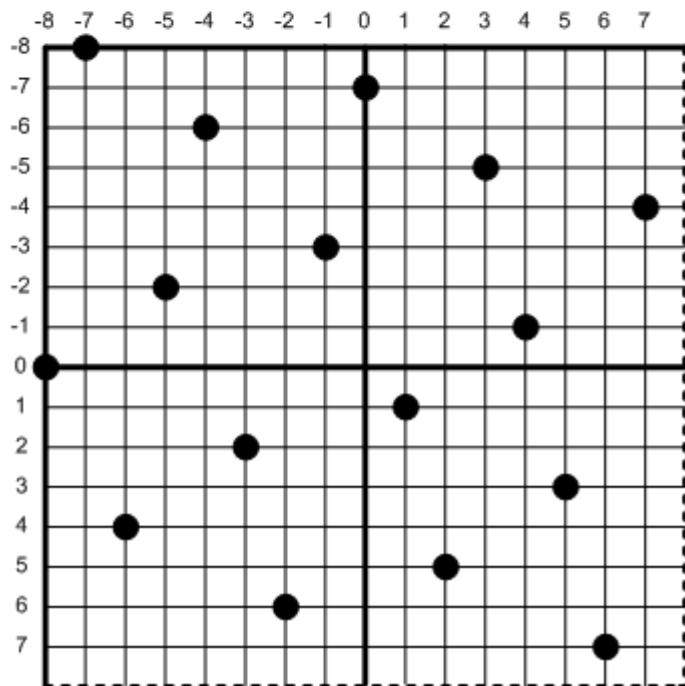
● { (-2,-6),(6,-2),(-6,2),(2,6) }

## Standard 8 Sample Pattern



● { (1,-3), (-1,3), (5,1), (-3,-5),  
(-5,5), (-7,-1), (3,7), (7,-7) }

## Standard 16 Sample Pattern



● { (1,1), (-1,-3), (-3,2), (4,-1),  
(-5,-2), (2,5), (5,3), (3,-5),  
(-2,6), (0,-7), (-4,-6), (-6,4),  
(-8,0), (7,-4), (6,7), (-7,-8) }

## Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_TEXTURE\_LAYOUT enumeration (d3d11\_3.h)

Article07/27/2022

Specifies texture layout options.

## Syntax

C++

```
typedef enum D3D11_TEXTURE_LAYOUT {
    D3D11_TEXTURE_LAYOUT_UNDEFINED = 0,
    D3D11_TEXTURE_LAYOUT_ROW_MAJOR = 1,
    D3D11_TEXTURE_LAYOUT_64K_STANDARD_SWIZZLE = 2
};
```

## Constants

`D3D11_TEXTURE_LAYOUT_UNDEFINED`

Value: 0

The texture layout is undefined, and is selected by the driver.

`D3D11_TEXTURE_LAYOUT_ROW_MAJOR`

Value: 1

Data for the texture is stored in row major (sometimes called pitch-linear) order.

`D3D11_TEXTURE_LAYOUT_64K_STANDARD_SWIZZLE`

Value: 2

A default texture uses the standardized swizzle pattern.

## Remarks

This enumeration controls the swizzle pattern of default textures and enable map support on default textures. Callers must query [D3D11\\_FEATURE\\_DATA\\_D3D11\\_OPTIONS2](#) to ensure that each option is supported.

The standard swizzle formats applies within each page-sized chunk, and pages are laid out in linear order with respect to one another. A 16-bit interleave pattern defines the conversion from pre-swizzled intra-page location to the post-swizzled location.

## 2D Texture Standard Swizzle Pattern

8bpp	XYXY XYXY YYYY XXXX		
16bpp	XYXY XYXY YYYY XXX-	32bpp	XYXY XYXY YYYY XX--
64bpp	XYXY XYXY XXYY X---	128bpp	XYXY XYXY XXYY ----

## 3D Texture Standard Swizzle Pattern

8bpp	XYZX YYZZ ZZYY XXXX		
16bpp	XYZX YYZZ ZZYY XXX-	32bpp	XYZX YZXY ZZYY XX--
64bpp	XYZX YZXX ZZYY X---	128bpp	XYZX YZXX ZZYY ----

To demonstrate, consider the 32bpp swizzle format above. This is represented by the following interleave masks, where bits on the left are most-significant.

### syntax

```
UINT xBytesMask = 1010 1010 1000 1111  
UINT yMask = 0101 0101 0111 0000
```

To compute the swizzled address, the following code could be used (where the `_pdep_u32` instruction is supported):

### syntax

```
UINT swizzledOffset = resourceBaseOffset +  
    _pdep_u32(xOffset, xBytesMask) +  
    _pdep_u32(yOffset, yBytesMask);
```

# Requirements

Header	d3d11_3.h
--------	-----------

# See also

[Resource Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3D11\_TILE\_COPY\_FLAG enumeration (d3d11\_2.h)

Article02/22/2024

Identifies how to copy a tile.

## Syntax

C++

```
typedef enum D3D11_TILE_COPY_FLAG {
    D3D11_TILE_COPY_NO_OVERWRITE = 0x1,
    D3D11_TILE_COPY_LINEAR_BUFFER_TO_SWIZZLED_TILED_RESOURCE = 0x2,
    D3D11_TILE_COPY_SWIZZLED_TILED_RESOURCE_TO_LINEAR_BUFFER = 0x4
} ;
```

## Constants

[+] Expand table

D3D11\_TILE\_COPY\_NO\_OVERWRITE

Value: 0x1

Indicates that the GPU isn't currently referencing any of the portions of destination memory being written.

D3D11\_TILE\_COPY\_LINEAR\_BUFFER\_TO\_SWIZZLED\_TILED\_RESOURCE

Value: 0x2

Indicates that the [ID3D11DeviceContext2::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 [ID3D11DeviceContext2::CopyTiles](#) call, you specify the source data with the *pBuffer* parameter and the destination with the *pTiledResource* parameter.

D3D11\_TILE\_COPY\_SWIZZLED\_TILED\_RESOURCE\_TO\_LINEAR\_BUFFER

Value: 0x4

Indicates that the [ID3D11DeviceContext2::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 `ID3D11DeviceContext2::CopyTiles` call, you specify the source data with the *pTiledResource* parameter and the destination with the *pBuffer* parameter.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11_2.h

## See also

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# D3D11\_TILE\_MAPPING\_FLAG enumeration (d3d11\_2.h)

Article02/22/2024

Identifies how to perform a tile-mapping operation.

## Syntax

C++

```
typedef enum D3D11_TILE_MAPPING_FLAG {
    D3D11_TILE_MAPPING_NO_OVERWRITE = 0x1
} ;
```

## Constants

[+] Expand table

D3D11\_TILE\_MAPPING\_NO\_OVERWRITE

Value: 0x1

Indicates that no overwriting of tiles occurs in the tile-mapping operation.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps   UWP apps]
Header	d3d11_2.h

## See also

[Resource Enumerations](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_TILE\_RANGE\_FLAG enumeration (d3d11\_2.h)

Article02/22/2024

Specifies a range of tile mappings to use with [ID3D11DeviceContext2::UpdateTiles](#).

## Syntax

C++

```
typedef enum D3D11_TILE_RANGE_FLAG {
    D3D11_TILE_RANGE_NULL = 0x1,
    D3D11_TILE_RANGE_SKIP = 0x2,
    D3D11_TILE_RANGE_REUSE_SINGLE_TILE = 0x4
} ;
```

## Constants

[+] Expand table

<b>D3D11_TILE_RANGE_NULL</b> Value: <i>0x1</i> The tile range is <b>NULL</b> .
<b>D3D11_TILE_RANGE_SKIP</b> Value: <i>0x2</i> Skip the tile range.
<b>D3D11_TILE_RANGE_REUSE_SINGLE_TILE</b> Value: <i>0x4</i> Reuse a single tile in the tile range.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Header	d3d11_2.h

## See also

[ID3D11DeviceContext2::UpdateTiles](#)

[Resource Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_UAV\_DIMENSION enumeration (d3d11.h)

Article02/22/2024

Unordered-access view options.

## Syntax

C++

```
typedef enum D3D11_UAV_DIMENSION {
    D3D11_UAV_DIMENSION_UNKNOWN = 0,
    D3D11_UAV_DIMENSION_BUFFER = 1,
    D3D11_UAV_DIMENSION_TEXTURE1D = 2,
    D3D11_UAV_DIMENSION_TEXTURE1DARRAY = 3,
    D3D11_UAV_DIMENSION_TEXTURE2D = 4,
    D3D11_UAV_DIMENSION_TEXTURE2DARRAY = 5,
    D3D11_UAV_DIMENSION_TEXTURE3D = 8
} ;
```

## Constants

[ ] Expand table

	<b>D3D11_UAV_DIMENSION_UNKNOWN</b>
Value:	0
	The view type is unknown.
	<b>D3D11_UAV_DIMENSION_BUFFER</b>
Value:	1
	View the resource as a buffer.
	<b>D3D11_UAV_DIMENSION_TEXTURE1D</b>
Value:	2
	View the resource as a 1D texture.
	<b>D3D11_UAV_DIMENSION_TEXTURE1DARRAY</b>
Value:	3
	View the resource as a 1D texture array.
	<b>D3D11_UAV_DIMENSION_TEXTURE2D</b>
Value:	4

View the resource as a 2D texture.

D3D11\_UAV\_DIMENSION\_TEXTURE2DARRAY

Value: 5

View the resource as a 2D texture array.

D3D11\_UAV\_DIMENSION\_TEXTURE3D

Value: 8

View the resource as a 3D texture array.

## Remarks

This enumeration is used by a unordered access-view description (see [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11.h

## See also

[Core Enumerations](#)

[Resource Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# D3D11\_USAGE enumeration (d3d11.h)

Article01/31/2022

Identifies expected resource use during rendering. The usage directly reflects whether a resource is accessible by the CPU and/or the graphics processing unit (GPU).

## Syntax

C++

```
typedef enum D3D11_USAGE {
    D3D11_USAGE_DEFAULT = 0,
    D3D11_USAGE_IMMUTABLE = 1,
    D3D11_USAGE_DYNAMIC = 2,
    D3D11_USAGE_STAGING = 3
} ;
```

## Constants

`D3D11_USAGE_DEFAULT`

Value: 0

A resource that requires read and write access by the GPU. This is likely to be the most common usage choice.

`D3D11_USAGE_IMMUTABLE`

Value: 1

A resource that can only be read by the GPU. It cannot be written by the GPU, and cannot be accessed at all by the CPU. This type of resource must be initialized when it is created, since it cannot be changed after creation.

`D3D11_USAGE_DYNAMIC`

Value: 2

A resource that is accessible by both the GPU (read only) and the CPU (write only). A dynamic resource is a good choice for a resource that will be updated by the CPU at least once per frame. To update a dynamic resource, use a `Map` method.

For info about how to use dynamic resources, see [How to: Use dynamic resources](#).

`D3D11_USAGE_STAGING`

Value: 3

A resource that supports data transfer (copy) from the GPU to the CPU.

## Remarks

An application identifies the way a resource is intended to be used (its usage) in a resource description. There are several structures for creating resources including: [D3D11\\_TEXTURE1D\\_DESC](#), [D3D11\\_TEXTURE2D\\_DESC](#), [D3D11\\_TEXTURE3D\\_DESC](#), and [D3D11\\_BUFFER\\_DESC](#).

Differences between Direct3D 9 and Direct3D 10/11:

In Direct3D 9, you specify the type of memory a resource should be created in at resource creation time (using D3DPOOL). It was an application's job to decide what memory pool would provide the best combination of functionality and performance.

In Direct3D 10/11, an application no longer specifies what type of memory (the pool) to create a resource in. Instead, you specify the intended usage of the resource, and let the runtime (in concert with the driver and a memory manager) choose the type of memory that will achieve the best performance.

## Resource Usage Restrictions

Each usage dictates a tradeoff between accessibility for the CPU and accessibility for the GPU. In general, higher-performance access for one of these two processors means lower-performance access for the other. At either extreme are the

[D3D11\\_USAGE\\_DEFAULT](#) and [D3D11\\_USAGE\\_STAGING](#) usages. [D3D11\\_USAGE\\_DEFAULT](#) restricts access almost entirely to the GPU. [D3D11\\_USAGE\\_STAGING](#) restricts access almost entirely to the CPU and allows only a data transfer (copy) of a resource between the GPU and the CPU. You can perform these copy operations via the [ID3D11DeviceContext::CopySubresourceRegion](#) and [ID3D11DeviceContext::CopyResource](#) methods. You can also use these copy methods to copy data between two resources of the same usage. You can also use the [ID3D11DeviceContext::UpdateSubresource](#) method to copy memory directly from a CPU-supplied pointer to any resource, most usefully a resource with [D3D11\\_USAGE\\_DEFAULT](#).

[D3D11\\_USAGE\\_DYNAMIC](#) usage is a special case that optimizes the flow of data from CPU to GPU when the CPU generates that data on-the-fly and sends that data with high frequency. [D3D11\\_USAGE\\_DYNAMIC](#) is typically used on resources with vertex data and on constant buffers. Use the [ID3D11DeviceContext::Map](#) and [ID3D11DeviceContext::Unmap](#) methods to write data to these resources. To achieve the highest performance for data consumed serially, like vertex data, use the [D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#) and [D3D11\\_MAP\\_WRITE\\_DISCARD](#) sequence. For

more info about this sequence, see [Common Usage of D3D11\\_MAP\\_WRITE\\_DISCARD with D3D11\\_MAP\\_WRITE\\_NO\\_OVERWRITE](#).

**D3D11\_USAGE\_IMMUTABLE** usage is another special case that causes the GPU to generate data just once when you create a resource. **D3D11\_USAGE\_IMMUTABLE** is well-suited to data such as textures because such data is typically read into memory from some file format. Therefore, when you create a texture with **D3D11\_USAGE\_IMMUTABLE**, the GPU directly reads that texture into memory.

Use the following table to choose the usage that best describes how the resource will need to be accessed by the CPU and/or the GPU. Of course, there will be performance tradeoffs.

Resource Usage	Default	Dynamic	Immutable	Staging
GPU-Read	yes	yes	yes	yes <sup>1</sup>
GPU-Write	yes			yes <sup>1</sup>
CPU-Read				yes <sup>1</sup>
CPU-Write		yes		yes <sup>1</sup>

1 - GPU read or write of a resource with the **D3D11\_USAGE\_STAGING** usage is restricted to copy operations. You use [ID3D11DeviceContext::CopySubresourceRegion](#) and [ID3D11DeviceContext::CopyResource](#) for these copy operations. Also, because depth-stencil formats and multisample layouts are implementation details of a particular GPU design, the operating system can't expose these formats and layouts to the CPU in general. Therefore, staging resources can't be a depth-stencil buffer or a multisampled render target.

**Note** You can technically use [ID3D11DeviceContext::UpdateSubresource](#) to copy to a resource with any usage except **D3D11\_USAGE\_IMMUTABLE**. However, we recommend to use [ID3D11DeviceContext::UpdateSubresource](#) to update only a resource with **D3D11\_USAGE\_DEFAULT**. We recommend to use [ID3D11DeviceContext::Map](#) and [ID3D11DeviceContext::Unmap](#) to update resources with **D3D11\_USAGE\_DYNAMIC** because that is the specific purpose of **D3D11\_USAGE\_DYNAMIC** resources, and is therefore the most optimized path.

**Note** `D3D11_USAGE_DYNAMIC` resources consume specific hardware capabilities. Therefore, use them sparingly. The display driver typically allocates memory for `D3D11_USAGE_DYNAMIC` resources with a caching algorithm that favors CPU writes and hinders CPU reads. Furthermore, the memory behind `D3D11_USAGE_DYNAMIC` resources might not even be the same for successive calls to `ID3D11DeviceContext::Map`. Therefore, do not expect high performance or even consistent CPU reads from `D3D11_USAGE_DYNAMIC` resources.

**Note** `ID3D11DeviceContext::CopyStructureCount` is a special case of GPU-to-CPU copy. Use `ID3D11DeviceContext::CopyStructureCount` only with unordered access views (UAVs) of buffers.

## Resource Bind Options

To maximize performance, not all resource usage options can be used as input or output resources to the pipeline. This table identifies these limitations.

Resource Can Be Bound As	Default	Dynamic	Immutable	Staging
Input to a Stage	yes <sup>2</sup>	yes <sup>3</sup>	yes	
Output from a Stage	yes <sup>2</sup>			

- 2 - If bound as an input and an output using different views, each view must use different subresources.
- 3 - The resource can only be created with a single subresource. The resource cannot be a texture array. The resource cannot be a mipmap chain.

## Requirements

Header d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

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

# Shader Reference (Direct3D 11 Graphics)

Article • 12/10/2020

The Direct3D API defines several API elements to help you create and manage programmable shaders. Shaders are executable programs that are programmed exclusively using HLSL.

## In this section

Topic	Description
<a href="#">Shader Interfaces</a>	This section contains information about the shader interfaces.
<a href="#">Shader Functions</a>	This section contains information about the shader functions.
<a href="#">Shader Structures</a>	Structures are used to create and use shaders.
<a href="#">Shader Enumerations</a>	Enumerations are used to specify information about shaders.

## Related topics

[Direct3D 11 Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Shader Interfaces (Direct3D 11 Graphics)

Article • 08/18/2023

This section contains information about the shader interfaces.

Each of these shader interfaces manages a compiled shader. The interface is created when a shader is compiled, and is then passed to various APIs that need access to a compiled shader; such as when binding a shader to a pipeline stage or getting a shader signature.

## In this section

Topic	Description
<a href="#">ID3D11ClassInstance</a>	This interface encapsulates an HLSL class.
<a href="#">ID3D11ClassLinkage</a>	This interface encapsulates an HLSL dynamic linkage.
<a href="#">ID3D11ComputeShader</a>	A compute-shader interface manages an executable program (a compute shader) that controls the compute-shader stage.
<a href="#">ID3D11DomainShader</a>	A domain-shader interface manages an executable program (a domain shader) that controls the domain-shader stage.
<a href="#">ID3D11FunctionLinkingGraph</a>	A function-linking-graph interface is used for constructing shaders that consist of a sequence of precompiled function calls that pass values to each other.  <b>Note:</b> This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.
<a href="#">ID3D11FunctionReflection</a>	A function-reflection interface accesses function info.  <b>Note:</b> This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.
<a href="#">ID3D11FunctionParameterReflection</a>	A function-parameter-reflection interface accesses function-parameter info.  <b>Note:</b> This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11

Topic	Description
	<p>platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.</p>
<a href="#">ID3D11GeometryShader</a>	<p>A geometry-shader interface manages an executable program (a geometry shader) that controls the geometry-shader stage.</p>
<a href="#">ID3D11HullShader</a>	<p>A hull-shader interface manages an executable program (a hull shader) that controls the hull-shader stage.</p>
<a href="#">ID3D11LibraryReflection</a>	<p>A library-reflection interface accesses library info.  <b>Note:</b> This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.</p>
<a href="#">ID3D11Linker</a>	<p>A linker interface is used to link a shader module.  <b>Note:</b> This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.</p>
<a href="#">ID3D11LinkingNode</a>	<p>A linking-node interface is used for shader linking.  <b>Note:</b> This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.</p>
<a href="#">ID3D11Module</a>	<p>A module interface creates an instance of a module that is used for resource rebinding.  <b>Note:</b> This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.</p>
<a href="#">ID3D11ModuleInstance</a>	<p>A module-instance interface is used for resource rebinding.  <b>Note:</b> This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.</p>

Topic	Description
<a href="#">ID3D11PixelShader</a>	A pixel-shader interface manages an executable program (a pixel shader) that controls the pixel-shader stage.
<a href="#">ID3D11ShaderReflection</a>	A shader-reflection interface accesses shader information.
<a href="#">ID3D11ShaderReflectionConstantBuffer</a>	This shader-reflection interface provides access to a constant buffer.
<a href="#">ID3D11ShaderReflectionType</a>	This shader-reflection interface provides access to variable type.
<a href="#">ID3D11ShaderReflectionVariable</a>	This shader-reflection interface provides access to a variable.
<a href="#">ID3D11ShaderTrace</a>	An <a href="#">ID3D11ShaderTrace</a> interface implements methods for obtaining traces of shader executions.
<a href="#">ID3D11ShaderTraceFactory</a>	An <a href="#">ID3D11ShaderTraceFactory</a> interface implements a method for generating shader trace information objects.
<a href="#">ID3D11VertexShader</a>	A vertex-shader interface manages an executable program (a vertex shader) that controls the vertex-shader stage.

## Related topics

[Shader Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ClassInstance interface (d3d11.h)

Article 02/22/2024

This interface encapsulates an HLSL class.

## Inheritance

The **ID3D11ClassInstance** interface inherits from [ID3D11DeviceChild](#).

**ID3D11ClassInstance** also has these types of members:

## Methods

The **ID3D11ClassInstance** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11ClassInstance::GetClassLinkage</a>
Gets the ID3D11ClassLinkage object associated with the current HLSL class.
<a href="#">ID3D11ClassInstance::GetDesc</a>
Gets a description of the current HLSL class.
<a href="#">ID3D11ClassInstance::GetInstanceName</a>
Gets the instance name of the current HLSL class.
<a href="#">ID3D11ClassInstance::GetType Name</a>
Gets the type of the current HLSL class.

## Remarks

This interface is created by calling [ID3D11ClassLinkage::CreateClassInstance](#). The interface is used when binding shader resources to the pipeline using APIs such as [ID3D11DeviceContext::VSSetShader](#).

## Requirements

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?



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

# ID3D11ClassInstance::GetClassLinkage method (d3d11.h)

Article02/22/2024

Gets the [ID3D11ClassLinkage](#) object associated with the current HLSL class.

## Syntax

C++

```
void GetClassLinkage(  
    [out] ID3D11ClassLinkage **ppLinkage  
>;
```

## Parameters

[out] ppLinkage

Type: [ID3D11ClassLinkage\\*\\*](#)

A pointer to a [ID3D11ClassLinkage](#) interface pointer.

## Return value

None

## Remarks

For more information about using the [ID3D11ClassInstance](#) interface, see [Dynamic Linking](#).

**Windows Phone 8:** This API is supported.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11ClassInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ClassInstance::GetDesc method (d3d11.h)

Article 10/13/2021

Gets a description of the current HLSL class.

## Syntax

C++

```
void GetDesc(  
    [out] D3D11_CLASS_INSTANCE_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_CLASS\\_INSTANCE\\_DESC\\*](#)

A pointer to a [D3D11\\_CLASS\\_INSTANCE\\_DESC](#) structure that describes the current HLSL class.

## Return value

None

## Remarks

For more information about using the [ID3D11ClassInstance](#) interface, see [Dynamic Linking](#).

An instance is not restricted to being used for a single type in a single shader. An instance is flexible and can be used for any shader that used the same type name or instance name when the instance was generated.

- A created instance will work for any shader that contains a type of the same type name. For instance, a class instance created with the type name **DefaultShader**

would work in any shader that contained a type `DefaultShader` even though several shaders could describe a different type.

- A gotten instance maps directly to an instance name/index in a shader. A class instance acquired using `GetClassInstance` will work for any shader that contains a class instance of the name used to generate the runtime instance, the instance does not have to be the same type in all of the shaders it's used in.

An instance does not replace the importance of reflection for a particular shader since a gotten instance will not know its slot location and a created instance only specifies a type name.

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11ClassInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ClassInstance::GetInstanceName method (d3d11.h)

Article 02/22/2024

Gets the instance name of the current HLSL class.

## Syntax

C++

```
void GetInstanceName(
    [out, optional] LPSTR pInstanceName,
    [in, out]        SIZE_T *pBufferLength
);
```

## Parameters

[out, optional] *pInstanceName*

Type: [LPSTR](#)

The instance name of the current HLSL class.

[in, out] *pBufferLength*

Type: [SIZE\\_T\\*](#)

The length of the *pInstanceName* parameter.

## Return value

None

## Remarks

`GetInstanceName` will return a valid name only for instances acquired using [ID3D11ClassLinkage::GetClassInstance](#).

For more information about using the `ID3D11ClassInstance` interface, see [Dynamic Linking](#).

Windows Phone 8: This API is supported.

# Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11ClassInstance](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ClassInstance::GetTypeName method (d3d11.h)

Article 02/22/2024

Gets the type of the current HLSL class.

## Syntax

C++

```
void GetTypeName(  
    [out, optional] LPSTR pTypeName,  
    [in, out]        SIZE_T *pBufferLength  
);
```

## Parameters

[out, optional] *pTypeName*

Type: [LPSTR](#)

Type of the current HLSL class.

[in, out] *pBufferLength*

Type: [SIZE\\_T\\*](#)

The length of the *pTypeName* parameter.

## Return value

None

## Remarks

`GetTypeName` will return a valid name only for instances acquired using [ID3D11ClassLinkage::GetClassInstance](#).

For more information about using the `ID3D11ClassInstance` interface, see [Dynamic Linking](#).

Windows Phone 8: This API is supported.

# Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3d11.lib

## See also

[ID3D11ClassInstance](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ClassLinkage interface (d3d11.h)

Article 02/22/2024

This interface encapsulates an HLSL dynamic linkage.

## Inheritance

The **ID3D11ClassLinkage** interface inherits from [ID3D11DeviceChild](#). **ID3D11ClassLinkage** also has these types of members:

## Methods

The **ID3D11ClassLinkage** interface has these methods.

[+] [Expand table](#)

<a href="#">ID3D11ClassLinkage::CreateClassInstance</a>
Initializes a class-instance object that represents an HLSL class instance.
<a href="#">ID3D11ClassLinkage::GetClassInstance</a>
Gets the class-instance object that represents the specified HLSL class.

## Remarks

A class linkage object can hold up to 64K gotten instances. A gotten instance is a handle that references a variable name in any shader that is created with that linkage object. When you create a shader with a class linkage object, the runtime gathers these instances and stores them in the class linkage object. For more information about how a class linkage object is used, see [Storing Variables and Types for Shaders to Share](#).

An **ID3D11ClassLinkage** object is created using the [ID3D11Device::CreateClassLinkage](#) method.

## Requirements

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[Core Interfaces](#)

[ID3D11DeviceChild](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# ID3D11ClassLinkage::CreateClassInstance method (d3d11.h)

Article 10/13/2021

Initializes a class-instance object that represents an HLSL class instance.

## Syntax

C++

```
HRESULT CreateClassInstance(
    [in]    LPCSTR          pClassName,
    [in]    UINT             ConstantBufferOffset,
    [in]    UINT             ConstantVectorOffset,
    [in]    UINT             TextureOffset,
    [in]    UINT             SamplerOffset,
    [out]   ID3D11ClassInstance **ppInstance
);
```

## Parameters

[in] `pClassName`

Type: [LPCSTR](#)

The type name of a class to initialize.

[in] `ConstantBufferOffset`

Type: [UINT](#)

Identifies the constant buffer that contains the class data.

[in] `ConstantVectorOffset`

Type: [UINT](#)

The four-component vector offset from the start of the constant buffer where the class data will begin. Consequently, this is not a byte offset.

[in] `TextureOffset`

Type: [UINT](#)

The texture slot for the first texture; there may be multiple textures following the offset.

[in] SamplerOffset

Type: [UINT](#)

The sampler slot for the first sampler; there may be multiple samplers following the offset.

[out] ppInstance

Type: [ID3D11ClassInstance\\*\\*](#)

The address of a pointer to an [ID3D11ClassInstance](#) interface to initialize.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Instances can be created (or gotten) before or after a shader is created. Use the same shader linkage object to acquire a class instance and create the shader the instance is going to be used in.

For more information about using the [ID3D11ClassLinkage](#) interface, see [Dynamic Linking](#).

**Windows Phone 8:** This API is supported.

## Requirements

Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11ClassLinkage](#)

---

## Feedback

Was this page helpful?



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

# ID3D11ClassLinkage::GetClassInstance method (d3d11.h)

Article 02/22/2024

Gets the class-instance object that represents the specified HLSL class.

## Syntax

C++

```
HRESULT GetClassInstance(
    [in]    LPCSTR             pClassName,
    [in]    UINT               InstanceIndex,
    [out]   ID3D11ClassInstance **ppInstance
);
```

## Parameters

[in] pClassName

Type: [LPCSTR](#)

The name of a class for which to get the class instance.

[in] InstanceIndex

Type: [UINT](#)

The index of the class instance.

[out] ppInstance

Type: [ID3D11ClassInstance\\*\\*](#)

The address of a pointer to an [ID3D11ClassInstance](#) interface to initialize.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

# Remarks

For more information about using the [ID3D11ClassLinkage](#) interface, see [Dynamic Linking](#).

A class instance must have at least 1 data member in order to be available for the runtime to use with [ID3D11ClassLinkage::GetClassInstance](#). Any instance with no members will be optimized out of a compiled shader blob as a zero-sized object. If you have a class with no data members, use [ID3D11ClassLinkage::CreateClassInstance](#) instead.

**Windows Phone 8:** This API is supported.

# Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[ID3D11ClassInstance](#)

[ID3D11ClassLinkage](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11ComputeShader interface (d3d11.h)

Article 02/16/2023

A compute-shader interface manages an executable program (a compute shader) that controls the compute-shader stage.

## Inheritance

The [ID3D11ComputeShader](#) interface inherits from the [ID3D11DeviceChild](#) interface.

## Remarks

The compute-shader interface has no methods; use HLSL to implement your shader functionality. All shaders are implemented from a common set of features referred to as the common-shader core..

To create a compute-shader interface, call [ID3D11Device::CreateComputeShader](#). Before using a compute shader you must bind it to the device by calling [ID3D11DeviceContext::CSSetShader](#).

This interface is defined in D3D11.h.

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11DeviceChild](#)

[Shader Interfaces](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11DomainShader interface (d3d11.h)

Article 02/22/2024

A domain-shader interface manages an executable program (a domain shader) that controls the domain-shader stage.

## Inheritance

The **ID3D11DomainShader** interface inherits from the **ID3D11DeviceChild** interface.

## Remarks

The domain-shader interface has no methods; use HLSL to implement your shader functionality. All shaders are implemented from a common set of features referred to as the common-shader core..

To create a domain-shader interface, call [ID3D11Device::CreateDomainShader](#). Before using a domain shader you must bind it to the device by calling [ID3D11DeviceContext::DSSetShader](#).

This interface is defined in D3D11.h.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11DeviceChild](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionLinkingGraph interface (d3d11shader.h)

Article 10/05/2021

A function-linking-graph interface is used for constructing shaders that consist of a sequence of precompiled function calls that pass values to each other.

**Note** This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

## Inheritance

The **ID3D11FunctionLinkingGraph** interface inherits from the [IUnknown](#) interface.  
**ID3D11FunctionLinkingGraph** also has these types of members:

## Methods

The **ID3D11FunctionLinkingGraph** interface has these methods.

### [ID3D11FunctionLinkingGraph::CallFunction](#)

Creates a call-function linking node to use in the function-linking-graph.

### [ID3D11FunctionLinkingGraph::CreateModuleInstance](#)

Initializes a shader module from the function-linking-graph object.

### [ID3D11FunctionLinkingGraph::GenerateHlsl](#)

Generates Microsoft High Level Shader Language (HLSL) shader code that represents the function-linking-graph.

### [ID3D11FunctionLinkingGraph::GetLastError](#)

Gets the error from the last function call of the function-linking-graph.

#### [ID3D11FunctionLinkingGraph::PassValue](#)

Passes a value from a source linking node to a destination linking node.

#### [ID3D11FunctionLinkingGraph::PassValueWithSwizzle](#)

Passes a value with swizzle from a source linking node to a destination linking node.

#### [ID3D11FunctionLinkingGraph::SetInputSignature](#)

Sets the input signature of the function-linking-graph.

#### [ID3D11FunctionLinkingGraph::SetOutputSignature](#)

Sets the output signature of the function-linking-graph.

## Remarks

To get a function-linking-graph interface, call [D3DCreateFunctionLinkingGraph](#).

You can use the function-linking-graph (FLG) interface methods to construct shaders that consist of a sequence of precompiled function calls that pass values to each other. You don't need to write HLSL and then call the HLSL compiler. Instead, the shader structure is specified programmatically via a C++ API. FLG nodes represent input and output signatures and invocations of precompiled library functions. The order of registering the function-call nodes defines the sequence of invocations. You must specify the input signature node first and the output signature node last. FLG edges define how values are passed from one node to another. The data types of passed values must be the same; there is no implicit type conversion. Shape and swizzling rules follow the HLSL behavior. Values can only be passed forward in this sequence.

**Note** **ID3D11FunctionLinkingGraph** requires the D3dcompiler\_47.dll or a later version of the DLL.

## Requirements

Target Platform

Windows

## See also

[IUnknown](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionLinkingGraph::CallFunction method (d3d11shader.h)

Article 02/22/2024

Creates a call-function linking node to use in the function-linking-graph.

## Syntax

C++

```
HRESULT CallFunction(
    [in, optional] LPCSTR           pModuleInstanceNamespace,
    [in]          ID3D11Module*      pModuleWithFunctionPrototype,
    [in]          LPCSTR           pFunctionName,
    [out]         ID3D11LinkingNode** ppCallNode
);
```

## Parameters

[in, optional] pModuleInstanceNamespace

Type: [LPCSTR](#)

The optional namespace for the function, or **NULL** if no namespace is needed.

[in] pModuleWithFunctionPrototype

Type: [ID3D11Module\\*](#)

A pointer to the [ID3D11ModuleInstance](#) interface for the library module that contains the function prototype.

[in] pFunctionName

Type: [LPCSTR](#)

The name of the function.

[out] ppCallNode

Type: [ID3D11LinkingNode\\*\\*](#)

A pointer to a variable that receives a pointer to the [ID3D11LinkingNode](#) interface that represents the function in the function-linking-graph.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

  [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionLinkingGraph](#)

---

## Feedback

Was this page helpful?



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

# ID3D11FunctionLinkingGraph::CreateModuleInstance method (d3d11shader.h)

Article 02/22/2024

Initializes a shader module from the function-linking-graph object.

## Syntax

C++

```
HRESULT CreateModuleInstance(
    [out]           ID3D11ModuleInstance **ppModuleInstance,
    [out, optional] ID3DBlob          **ppErrorBuffer
);
```

## Parameters

[out] ppModuleInstance

Type: [ID3D11ModuleInstance\\*\\*](#)

The address of a pointer to an [ID3D11ModuleInstance](#) interface for the shader module to initialize.

[out, optional] ppErrorBuffer

Type: [ID3DBlob\\*\\*](#)

An optional pointer to a variable that receives a pointer to the [ID3DBlob](#) interface that you can use to access compiler error messages, or [NULL](#) if there are no errors.

## Return value

Type: [HRESULT](#)

Returns [S\\_OK](#) if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionLinkingGraph](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionLinkingGraph::GenerateHlsl method (d3d11shader.h)

Article 02/22/2024

Generates Microsoft High Level Shader Language (HLSL) shader code that represents the function-linking-graph.

## Syntax

C++

```
HRESULT GenerateHlsl(
    [in]  UINT      uFlags,
    [out] ID3DBlob **ppBuffer
);
```

## Parameters

[in] uFlags

Type: [UINT](#)

Reserved

[out] ppBuffer

Type: [ID3DBlob\\*\\*](#)

An pointer to a variable that receives a pointer to the [ID3DBlob](#) interface that you can use to access the HLSL shader source code that represents the function-linking-graph. You can compile this HLSL code, but first you must add code or include statements for the functions called in the function-linking-graph.

## Return value

Type: [HRESULT](#)

Returns [S\\_OK](#) if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionLinkingGraph](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionLinkingGraph::GetLastErr or method (d3d11shader.h)

Article 02/22/2024

Gets the error from the last function call of the function-linking-graph.

## Syntax

C++

```
HRESULT GetLastErrorMessage(
    [out, optional] ID3DBlob **ppErrorBuffer
);
```

## Parameters

[out, optional] ppErrorBuffer

Type: [ID3DBlob\\*\\*](#)

An pointer to a variable that receives a pointer to the [ID3DBlob](#) interface that you can use to access the error.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib

Requirement	Value
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionLinkingGraph](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionLinkingGraph::PassValue method (d3d11shader.h)

Article 02/22/2024

Passes a value from a source linking node to a destination linking node.

## Syntax

C++

```
HRESULT PassValue(
    [in] ID3D11LinkingNode *pSrcNode,
    [in] INT             SrcParameterIndex,
    [in] ID3D11LinkingNode *pDstNode,
    [in] INT             DstParameterIndex
);
```

## Parameters

[in] pSrcNode

Type: [ID3D11LinkingNode\\*](#)

A pointer to the [ID3D11LinkingNode](#) interface for the source linking node.

[in] SrcParameterIndex

Type: INT

The zero-based index of the source parameter.

[in] pDstNode

Type: [ID3D11LinkingNode\\*](#)

A pointer to the [ID3D11LinkingNode](#) interface for the destination linking node.

[in] DstParameterIndex

Type: INT

The zero-based index of the destination parameter.

# Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

  [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionLinkingGraph](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionLinkingGraph::PassValueWithSwizzle method (d3d11shader.h)

Article 10/13/2021

Passes a value with swizzle from a source linking node to a destination linking node.

## Syntax

C++

```
HRESULT PassValueWithSwizzle(
    [in] ID3D11LinkingNode *pSrcNode,
    [in] INT             SrcParameterIndex,
    [in] LPCSTR          pSrcSwizzle,
    [in] ID3D11LinkingNode *pDstNode,
    [in] INT             DstParameterIndex,
    [in] LPCSTR          pDstSwizzle
);
```

## Parameters

[in] pSrcNode

Type: [ID3D11LinkingNode\\*](#)

A pointer to the [ID3D11LinkingNode](#) interface for the source linking node.

[in] SrcParameterIndex

Type: [INT](#)

The zero-based index of the source parameter.

[in] pSrcSwizzle

Type: [LPCSTR](#)

The name of the source swizzle.

[in] pDstNode

Type: [ID3D11LinkingNode\\*](#)

A pointer to the [ID3D11LinkingNode](#) interface for the destination linking node.

[in] DstParameterIndex

Type: **INT**

The zero-based index of the destination parameter.

[in] pDstSwizzle

Type: [LPCSTR](#)

The name of the destination swizzle.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionLinkingGraph](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionLinkingGraph::SetInputSignature method (d3d11shader.h)

Article 02/22/2024

Sets the input signature of the function-linking-graph.

## Syntax

C++

```
HRESULT SetInputSignature(  
    [in]  const D3D11_PARAMETER_DESC *pInputParameters,  
    [in]  UINT                  cInputParameters,  
    [out] ID3D11LinkingNode     **ppInputNode  
) ;
```

## Parameters

[in] pInputParameters

Type: **const D3D11\_PARAMETER\_DESC\***

An array of **D3D11\_PARAMETER\_DESC** structures for the parameters of the input signature.

[in] cInputParameters

Type: **UINT**

The number of input parameters in the *pInputParameters* array.

[out] ppInputNode

Type: **ID3D11LinkingNode\*\***

A pointer to a variable that receives a pointer to the **ID3D11LinkingNode** interface that represents the input signature of the function-linking-graph.

## Return value

Type: **HRESULT**

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

# Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionLinkingGraph](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionLinkingGraph::SetOutputSignature method (d3d11shader.h)

Article 02/22/2024

Sets the output signature of the function-linking-graph.

## Syntax

C++

```
HRESULT SetOutputSignature(
    [in]  const D3D11_PARAMETER_DESC *pOutputParameters,
    [in]  UINT                  cOutputParameters,
    [out] ID3D11LinkingNode     **ppOutputNode
);
```

## Parameters

[in] *pOutputParameters*

Type: **const D3D11\_PARAMETER\_DESC\***

An array of **D3D11\_PARAMETER\_DESC** structures for the parameters of the output signature.

[in] *cOutputParameters*

Type: **UINT**

The number of output parameters in the *pOutputParameters* array.

[out] *ppOutputNode*

Type: **ID3D11LinkingNode\*\***

A pointer to a variable that receives a pointer to the **ID3D11LinkingNode** interface that represents the output signature of the function-linking-graph.

## Return value

Type: **HRESULT**

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

# Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionLinkingGraph](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionReflection interface (d3d11shader.h)

Article 02/22/2024

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 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

## Methods

The **ID3D11FunctionReflection** interface has these methods.

[+] Expand table

<a href="#">ID3D11FunctionReflection::GetConstantBufferByIndex</a>
The <b>ID3D11FunctionReflection::GetConstantBufferByIndex</b> (d3d11shader.h) method gets a constant buffer by index for a function.
<a href="#">ID3D11FunctionReflection::GetConstantBufferByName</a>
Gets a constant buffer by name for a function. ( <b>ID3D11FunctionReflection.GetConstantBufferByName</b> )
<a href="#">ID3D11FunctionReflection::GetDesc</a>
Fills the function descriptor structure for the function. ( <b>ID3D11FunctionReflection.GetDesc</b> )
<a href="#">ID3D11FunctionReflection::GetFunctionParameter</a>
Gets the function parameter reflector. ( <b>ID3D11FunctionReflection.GetFunctionParameter</b> )
<a href="#">ID3D11FunctionReflection::GetResourceBindingDesc</a>
Gets a description of how a resource is bound to a function. ( <b>ID3D11FunctionReflection.GetResourceBindingDesc</b> )

### [ID3D11FunctionReflection::GetResourceBindingDescByName](#)

Gets a description of how a resource is bound to a function.  
(ID3D11FunctionReflection.GetResourceBindingDescByName)

### [ID3D11FunctionReflection::GetVariableByName](#)

Gets a variable by name. (ID3D11FunctionReflection.GetVariableByName)

## Remarks

To get a function-reflection interface, call [ID3D11LibraryReflection::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** `ID3D11FunctionReflection` requires the `D3dcompiler_47.dll` or a later version of the DLL.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	<code>d3d11shader.h</code>

## See also

[Shader Interfaces](#)

## Feedback

Was this page helpful?

 Yes

 No



# ID3D11FunctionReflection::GetConstantBufferByIndex method (d3d11shader.h)

Article 02/22/2024

Gets a constant buffer by index for a function.

## Syntax

C++

```
ID3D11ShaderReflectionConstantBuffer * GetConstantBufferByIndex(  
    [in] UINT BufferIndex  
);
```

## Parameters

[in] BufferIndex

Type: [UINT](#)

Zero-based index.

## Return value

Type: [ID3D11ShaderReflectionConstantBuffer\\*](#)

A pointer to a [ID3D11ShaderReflectionConstantBuffer](#) 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

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionReflection](#)

---

## Feedback

Was this page helpful?



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

# ID3D11FunctionReflection::GetConstantBufferByName method (d3d11shader.h)

Article 02/22/2024

Gets a constant buffer by name for a function.

## Syntax

C++

```
ID3D11ShaderReflectionConstantBuffer * GetConstantBufferByName(  
    [in] LPCSTR Name  
);
```

## Parameters

[in] Name

Type: [LPCSTR](#)

The constant-buffer name.

## Return value

Type: [ID3D11ShaderReflectionConstantBuffer\\*](#)

A pointer to a [ID3D11ShaderReflectionConstantBuffer](#) 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

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionReflection](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ID3D11FunctionReflection::GetDesc method (d3d11shader.h)

Article 02/22/2024

Fills the function descriptor structure for the function.

## Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D11_FUNCTION_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_FUNCTION\\_DESC\\*](#)

A pointer to a [D3D11\\_FUNCTION\\_DESC](#) structure that receives a description of the function.

## Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 11 Return Codes](#).

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib

Requirement	Value
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionReflection](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# ID3D11FunctionReflection::GetFunctionParameter method (d3d11shader.h)

Article 02/22/2024

Gets the function parameter reflector.

## Syntax

C++

```
ID3D11FunctionParameterReflection * GetFunctionParameter(
    [in] INT ParameterIndex
);
```

## Parameters

[in] ParameterIndex

Type: INT

The zero-based index of the function parameter reflector to retrieve.

## Return value

Type: [ID3D11FunctionParameterReflection\\*](#)

A pointer to a [ID3D11FunctionParameterReflection](#) interface that represents the function parameter reflector.

## Requirements

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

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib

Requirement	Value
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionReflection](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11FunctionReflection::GetResourceBindingDesc method (d3d11shader.h)

Article 02/22/2024

Gets a description of how a resource is bound to a function.

## Syntax

C++

```
HRESULT GetResourceBindingDesc(
    [in]  UINT             ResourceIndex,
    [out] D3D11_SHADER_INPUT_BIND_DESC *pDesc
);
```

## Parameters

[in] ResourceIndex

Type: [UINT](#)

A zero-based resource index.

[out] pDesc

Type: [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC\\*](#)

A pointer to a [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC](#) structure that describes input binding of the resource.

## Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 11 Return Codes](#).

## Remarks

A shader consists of executable code (the compiled HLSL functions) and a set of resources that supply the shader with input data. `GetResourceBindingDesc` gets info

about how one resource in the set is bound as an input to the shader. The *ResourceIndex* parameter specifies the index for the resource.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionReflection](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11FunctionReflection::GetResourceBindingDescByName method (d3d11shader.h)

Article 02/22/2024

Gets a description of how a resource is bound to a function.

## Syntax

C++

```
HRESULT GetResourceBindingDescByName(
    [in]    LPCSTR                 Name,
    [out]   D3D11_SHADER_INPUT_BIND_DESC *pDesc
);
```

## Parameters

[in] Name

Type: [LPCSTR](#)

The constant-buffer name of the resource.

[out] pDesc

Type: [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC\\*](#)

A pointer to a [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC](#) structure that describes input binding of the resource.

## Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 11 Return Codes](#).

## Remarks

A shader consists of executable code (the compiled HLSL functions) and a set of resources that supply the shader with input data. `GetResourceBindingDescByName` gets info about how one resource in the set is bound as an input to the shader. The `Name` parameter specifies the name of the resource.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionReflection](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11FunctionReflection::GetVariableByName method (d3d11shader.h)

Article 02/22/2024

Gets a variable by name.

## Syntax

C++

```
ID3D11ShaderReflectionVariable * GetVariableByName(  
    [in] LPCSTR Name  
) ;
```

## Parameters

[in] Name

Type: [LPCSTR](#)

A pointer to a string containing the variable name.

## Return value

Type: [ID3D11ShaderReflectionVariable\\*](#)

Returns a [ID3D11ShaderReflectionVariable Interface](#) interface.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib

Requirement	Value
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionReflection](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11FunctionParameterReflection interface (d3d11shader.h)

Article02/22/2024

A function-parameter-reflection interface accesses function-parameter info.

**Note** This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

## Methods

The **ID3D11FunctionParameterReflection** interface has these methods.

[+] Expand table

### [ID3D11FunctionParameterReflection::GetDesc](#)

Fills the parameter descriptor structure for the function's parameter.

([ID3D11FunctionParameterReflection.GetDesc](#))

## Remarks

To get a function-parameter-reflection interface, call

[ID3D11FunctionReflection::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** **ID3D11FunctionParameterReflection** requires the D3dcompiler\_47.dll or a later version of the DLL.

## Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h

## See also

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11FunctionParameterReflection::GetDesc method (d3d11shader.h)

Article 02/22/2024

Fills the parameter descriptor structure for the function's parameter.

## Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D11_PARAMETER_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_PARAMETER\\_DESC\\*](#)

A pointer to a [D3D11\\_PARAMETER\\_DESC](#) structure that receives a description of the function's parameter.

## Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 11 Return Codes](#).

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib

Requirement	Value
DLL	D3DCompiler_47.dll

## See also

[ID3D11FunctionParameterReflection](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11GeometryShader interface (d3d11.h)

Article 02/22/2024

A geometry-shader interface manages an executable program (a geometry shader) that controls the geometry-shader stage.

## Inheritance

The [ID3D11GeometryShader](#) interface inherits from the [ID3D11DeviceChild](#) interface.

## Remarks

The geometry-shader interface has no methods; use HLSL to implement your shader functionality. All shaders are implemented from a common set of features referred to as the common-shader core..

To create a geometry shader interface, call either [ID3D11Device::CreateGeometryShader](#) or [ID3D11Device::CreateGeometryShaderWithStreamOutput](#). Before using a geometry shader you must bind it to the device by calling [ID3D11DeviceContext::GSSetShader](#).

This interface is defined in D3D11.h.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11DeviceChild](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11HullShader interface (d3d11.h)

Article02/16/2023

A hull-shader interface manages an executable program (a hull shader) that controls the hull-shader stage.

## Inheritance

The **ID3D11HullShader** interface inherits from the **ID3D11DeviceChild** interface.

## Remarks

The hull-shader interface has no methods; use HLSL to implement your shader functionality. All shaders are implemented from a common set of features referred to as the common-shader core..

To create a hull-shader interface, call [ID3D11Device::CreateHullShader](#). Before using a hull shader you must bind it to the device by calling [ID3D11DeviceContext::HSSetShader](#).

This interface is defined in D3D11.h.

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11DeviceChild](#)

[Shader Interfaces](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11LibraryReflection interface (d3d11shader.h)

Article 02/22/2024

A library-reflection interface accesses library info.

**Note** This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

## Inheritance

The **ID3D11LibraryReflection** interface inherits from the [IUnknown](#) interface.  
**ID3D11LibraryReflection** also has these types of members:

## Methods

The **ID3D11LibraryReflection** interface has these methods.

[+] Expand table

<a href="#">ID3D11LibraryReflection::GetDesc</a>
Fills the library descriptor structure for the library reflection. ( <a href="#">ID3D11LibraryReflection.GetDesc</a> )
<a href="#">ID3D11LibraryReflection::GetFunctionByIndex</a>
The <a href="#">ID3D11LibraryReflection::GetFunctionByIndex</a> (d3d11shader.h) method gets the function reflector.

## Remarks

To get a library-reflection interface, call [D3DReflectLibrary](#).

**Note** ID3D11LibraryReflection requires the D3dcompiler\_47.dll or a later version of the DLL.

# Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h

## See also

[IUnknown](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11LibraryReflection::GetDesc method (d3d11shader.h)

Article 02/22/2024

Fills the library descriptor structure for the library reflection.

## Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D11_LIBRARY_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_LIBRARY\\_DESC\\*](#)

A pointer to a [D3D11\\_LIBRARY\\_DESC](#) structure that receives a description of the library reflection.

## Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 11 Return Codes](#).

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib

Requirement	Value
DLL	D3DCompiler_47.dll

## See also

[ID3D11LibraryReflection](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11LibraryReflection::GetFunctionByIndex method (d3d11shader.h)

Article 02/22/2024

Gets the function reflector.

## Syntax

C++

```
ID3D11FunctionReflection * GetFunctionByIndex(  
    [in] INT FunctionIndex  
);
```

## Parameters

[in] FunctionIndex

Type: INT

The zero-based index of the function reflector to retrieve.

## Return value

Type: [ID3D11FunctionReflection\\*](#)

A pointer to a [ID3D11FunctionReflection](#) interface that represents the function reflector.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib

Requirement	Value
DLL	D3DCompiler_47.dll

## See also

[ID3D11LibraryReflection](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11Linker interface (d3d11shader.h)

Article 02/22/2024

A linker interface is used to link a shader module.

**Note** This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

## Inheritance

The **ID3D11Linker** interface inherits from the [IUnknown](#) interface. **ID3D11Linker** also has these types of members:

## Methods

The **ID3D11Linker** interface has these methods.

[Expand table](#)

<a href="#">ID3D11Linker::AddClipPlaneFromCBuffer</a>
Adds a clip plane with the plane coefficients taken from a cbuffer entry for 10Level9 shaders.
<a href="#">ID3D11Linker::Link</a>
Links the shader and produces a shader blob that the Direct3D runtime can use.
<a href="#">ID3D11Linker::UseLibrary</a>
Adds an instance of a library module to be used for linking.

## Remarks

To get a linker interface, call [D3DCreateLinker](#).

**Note** ID3D11Linker requires the D3dcompiler\_47.dll or a later version of the DLL.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h

## See also

[IUnknown](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11Linker::AddClipPlaneFromCBuffer method (d3d11shader.h)

Article 02/22/2024

Adds a [clip plane](#) with the plane coefficients taken from a [cbuffer](#) entry for 10Level9 shaders.

## Syntax

C++

```
HRESULT AddClipPlaneFromCBuffer(
    [in] UINT uCBufferSlot,
    [in] UINT uCBufferEntry
);
```

## Parameters

`[in] uCBufferSlot`

Type: [UINT](#)

The [cbuffer](#) slot number.

`[in] uCBufferEntry`

Type: [UINT](#)

The [cbuffer](#) entry number.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11Linker](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11Linker::Link method (d3d11shader.h)

Article 02/22/2024

Links the shader and produces a shader blob that the Direct3D runtime can use.

## Syntax

C++

```
HRESULT Link(
    [in]          ID3D11ModuleInstance *pEntry,
    [in]          LPCSTR           pEntryName,
    [in]          LPCSTR           pTargetName,
    [in]          UINT             uFlags,
    [out]         ID3DBlob        **ppShaderBlob,
    [out, optional] ID3DBlob        **ppErrorBuffer
);
```

## Parameters

[in] pEntry

Type: [ID3D11ModuleInstance\\*](#)

A pointer to the [ID3D11ModuleInstance](#) interface for the shader module instance to link from.

[in] pEntryName

Type: [LPCSTR](#)

The name of the shader module instance to link from.

[in] pTargetName

Type: [LPCSTR](#)

The name for the shader blob that is produced.

[in] uFlags

Type: [UINT](#)

Reserved.

[out] ppShaderBlob

Type: [ID3DBlob\\*\\*](#)

A pointer to a variable that receives a pointer to the [ID3DBlob](#) interface that you can use to access the compiled shader code.

[out, optional] ppErrorBuffer

Type: [ID3DBlob\\*\\*](#)

A pointer to a variable that receives a pointer to the [ID3DBlob](#) interface that you can use to access compiler error messages.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11Linker](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11Linker::UseLibrary method (d3d11shader.h)

Article 02/22/2024

Adds an instance of a library module to be used for linking.

## Syntax

C++

```
HRESULT UseLibrary(  
    [in] ID3D11ModuleInstance *pLibraryMI  
);
```

## Parameters

[in] pLibraryMI

Type: [ID3D11ModuleInstance\\*](#)

A pointer to the [ID3D11ModuleInstance](#) interface for the library module instance.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib

Requirement	Value
DLL	D3DCompiler_47.dll

## See also

[ID3D11Linker](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11LinkingNode interface (d3d11shader.h)

Article 02/22/2024

A linking-node interface is used for shader linking.

**Note** This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

## Inheritance

The **ID3D11LinkingNode** interface inherits from the **IUnknown** interface.

## Remarks

To get a linking-node interface, call [ID3D11FunctionLinkingGraph::SetInputSignature](#), [ID3D11FunctionLinkingGraph::SetOutputSignature](#), or [ID3D11FunctionLinkingGraph::CallFunction](#).

**Note** **ID3D11LinkingNode** requires the D3dcompiler\_47.dll or a later version of the DLL.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h

## See also

[IUnknown](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11Module interface (d3d11shader.h)

Article 02/22/2024

A module interface creates an instance of a module that is used for resource rebinding.

**Note** This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

## Inheritance

The **ID3D11Module** interface inherits from the [IUnknown](#) interface. **ID3D11Module** also has these types of members:

## Methods

The **ID3D11Module** interface has these methods.

[Expand table](#)

### [ID3D11Module::CreateInstance](#)

Initializes an instance of a shader module that is used for resource rebinding.

## Remarks

To get a module interface, call [D3DLoadModule](#).

**Note** **ID3D11Module** requires the D3dcompiler\_47.dll or a later version of the DLL.

## Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h

## See also

[IUnknown](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11Module::CreateInstance method (d3d11shader.h)

Article 02/22/2024

Initializes an instance of a shader module that is used for resource rebinding.

## Syntax

C++

```
HRESULT CreateInstance(  
    [in, optional] LPCSTR pNamespace,  
    [out]           ID3D11ModuleInstance **ppModuleInstance  
) ;
```

## Parameters

[in, optional] pNamespace

Type: [LPCSTR](#)

The name of a shader module to initialize. This can be **NULL** if you don't want to specify a name for the module.

[out] ppModuleInstance

Type: [ID3D11ModuleInstance\\*\\*](#)

The address of a pointer to an [ID3D11ModuleInstance](#) interface to initialize.

## Return value

Type: [HRESULT](#)

Returns S\_OK if successful; otherwise, returns one of the [Direct3D 11 Return Codes](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11Module](#)

---

## Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance interface (d3d11shader.h)

Article 10/05/2021

A module-instance interface is used for resource rebinding.

**Note** This interface is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

## Inheritance

The **ID3D11ModuleInstance** interface inherits from the [IUnknown](#) interface.  
**ID3D11ModuleInstance** also has these types of members:

## Methods

The **ID3D11ModuleInstance** interface has these methods.

### [ID3D11ModuleInstance::BindConstantBuffer](#)

Rebinds a constant buffer from a source slot to a destination slot.

### [ID3D11ModuleInstance::BindConstantBufferByName](#)

Rebinds a constant buffer by name to a destination slot.

### [ID3D11ModuleInstance::BindResource](#)

Rebinds a texture or buffer from source slot to destination slot.

### [ID3D11ModuleInstance::BindResourceAsUnorderedAccessView](#)

Rebinds a resource as an unordered access view (UAV) from source slot to destination slot.

### [ID3D11ModuleInstance::BindResourceAsUnorderedAccessViewByName](#)

Rebinds a resource by name as an unordered access view (UAV) to destination slots.

#### [ID3D11ModuleInstance::BindResourceByName](#)

Rebinds a texture or buffer by name to destination slots.

#### [ID3D11ModuleInstance::BindSampler](#)

Rebinds a sampler from source slot to destination slot.

#### [ID3D11ModuleInstance::BindSamplerByName](#)

Rebinds a sampler by name to destination slots.

#### [ID3D11ModuleInstance::BindUnorderedAccessView](#)

Rebinds an unordered access view (UAV) from source slot to destination slot.

#### [ID3D11ModuleInstance::BindUnorderedAccessViewByName](#)

Rebinds an unordered access view (UAV) by name to destination slots.

## Remarks

To get a module-instance interface, call [ID3D11Module::CreateInstance](#) or [ID3D11FunctionLinkingGraph::CreateModuleInstance](#).

**Note** `ID3D11ModuleInstance` requires the `D3dcompiler_47.dll` or a later version of the DLL.

## Requirements

Target Platform	Windows
Header	<code>d3d11shader.h</code>

## See also

[IUnknown](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindConstantBuffer method (d3d11shader.h)

Article 02/22/2024

Rebinds a constant buffer from a source slot to a destination slot.

## Syntax

C++

```
HRESULT BindConstantBuffer(  
    [in] UINT uSrcSlot,  
    [in] UINT uDstSlot,  
    [in] UINT cbDstOffset  
);
```

## Parameters

[in] uSrcSlot

Type: [UINT](#)

The source slot number for rebinding.

[in] uDstSlot

Type: [UINT](#)

The destination slot number for rebinding.

[in] cbDstOffset

Type: [UINT](#)

The offset in bytes of the destination slot for rebinding. The offset must have 16-byte alignment.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindConstantBufferByName method (d3d11shader.h)

Article 02/22/2024

Rebinds a constant buffer by name to a destination slot.

## Syntax

C++

```
HRESULT BindConstantBufferByName(
    [in] LPCSTR pName,
    [in] UINT    uDstSlot,
    [in] UINT    cbDstOffset
);
```

## Parameters

[in] pName

Type: [LPCSTR](#)

The name of the constant buffer for rebinding.

[in] uDstSlot

Type: [UINT](#)

The destination slot number for rebinding.

[in] cbDstOffset

Type: [UINT](#)

The offset in bytes of the destination slot for rebinding. The offset must have 16-byte alignment.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindResource method (d3d11shader.h)

Article 02/22/2024

Rebinds a texture or buffer from source slot to destination slot.

## Syntax

C++

```
HRESULT BindResource(  
    [in] UINT uSrcSlot,  
    [in] UINT uDstSlot,  
    [in] UINT uCount  
);
```

## Parameters

[in] uSrcSlot

Type: [UINT](#)

The first source slot number for rebinding.

[in] uDstSlot

Type: [UINT](#)

The first destination slot number for rebinding.

[in] uCount

Type: [UINT](#)

The number of slots for rebinding.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindResourceAsUnorderedAccessView method (d3d11shader.h)

Article 10/13/2021

Rebinds a resource as an unordered access view (UAV) from source slot to destination slot.

## Syntax

C++

```
HRESULT BindResourceAsUnorderedAccessView(
    [in] UINT uSrcSrvSlot,
    [in] UINT uDstUavSlot,
    [in] UINT uCount
);
```

## Parameters

[in] uSrcSrvSlot

Type: **UINT**

The first source slot number for rebinding.

[in] uDstUavSlot

Type: **UINT**

The first destination slot number for rebinding.

[in] uCount

Type: **UINT**

The number of slots for rebinding.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindResourceAsUnorderedAccessViewByName method (d3d11shader.h)

Article 02/22/2024

Rebinds a resource by name as an unordered access view (UAV) to destination slots.

## Syntax

C++

```
HRESULT BindResourceAsUnorderedAccessViewByName(
    [in] LPCSTR pSrvName,
    [in] UINT    uDstUavSlot,
    [in] UINT    uCount
);
```

## Parameters

[in] pSrvName

Type: [LPCSTR](#)

The name of the resource for rebinding.

[in] uDstUavSlot

Type: [UINT](#)

The first destination slot number for rebinding.

[in] uCount

Type: [UINT](#)

The number of slots for rebinding.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindResourceByName method (d3d11shader.h)

Article 02/22/2024

Rebinds a texture or buffer by name to destination slots.

## Syntax

C++

```
HRESULT BindResourceByName(
    [in] LPCSTR pName,
    [in] UINT    uDstSlot,
    [in] UINT    uCount
);
```

## Parameters

[in] pName

Type: [LPCSTR](#)

The name of the texture or buffer for rebinding.

[in] uDstSlot

Type: [UINT](#)

The first destination slot number for rebinding.

[in] uCount

Type: [UINT](#)

The number of slots for rebinding.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindSampler method (d3d11shader.h)

Article 02/22/2024

Rebinds a sampler from source slot to destination slot.

## Syntax

C++

```
HRESULT BindSampler(  
    [in] UINT uSrcSlot,  
    [in] UINT uDstSlot,  
    [in] UINT uCount  
);
```

## Parameters

[in] uSrcSlot

Type: [UINT](#)

The first source slot number for rebinding.

[in] uDstSlot

Type: [UINT](#)

The first destination slot number for rebinding.

[in] uCount

Type: [UINT](#)

The number of slots for rebinding.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindSamplerByName method (d3d11shader.h)

Article 02/22/2024

Rebinds a sampler by name to destination slots.

## Syntax

C++

```
HRESULT BindSamplerByName(  
    [in] LPCSTR pName,  
    [in] UINT    uDstSlot,  
    [in] UINT    uCount  
>;
```

## Parameters

[in] pName

Type: [LPCSTR](#)

The name of the sampler for rebinding.

[in] uDstSlot

Type: [UINT](#)

The first destination slot number for rebinding.

[in] uCount

Type: [UINT](#)

The number of slots for rebinding.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindUnorderedAccessView method (d3d11shader.h)

Article 02/22/2024

Rebinds an unordered access view (UAV) from source slot to destination slot.

## Syntax

C++

```
HRESULT BindUnorderedAccessView(
    [in] UINT uSrcSlot,
    [in] UINT uDstSlot,
    [in] UINT uCount
);
```

## Parameters

[in] uSrcSlot

Type: [UINT](#)

The first source slot number for rebinding.

[in] uDstSlot

Type: [UINT](#)

The first destination slot number for rebinding.

[in] uCount

Type: [UINT](#)

The number of slots for rebinding.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ModuleInstance::BindUnorderedAccessViewByName method (d3d11shader.h)

Article 02/22/2024

Rebinds an unordered access view (UAV) by name to destination slots.

## Syntax

C++

```
HRESULT BindUnorderedAccessViewByName(
    [in] LPCSTR pName,
    [in] UINT    uDstSlot,
    [in] UINT    uCount
);
```

## Parameters

[in] pName

Type: [LPCSTR](#)

The name of the UAV for rebinding.

[in] uDstSlot

Type: [UINT](#)

The first destination slot number for rebinding.

[in] uCount

Type: [UINT](#)

The number of slots for rebinding.

## Return value

Type: [HRESULT](#)

Returns:

- **S\_OK** for a valid rebinding
- **S\_FALSE** for rebinding a nonexistent slot; that is, for which the shader reflection doesn't have any data
- **E\_FAIL** for an invalid rebinding, for example, the rebinding is out-of-bounds
- Possibly one of the other [Direct3D 11 Return Codes](#)

## Requirements

  [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ModuleInstance](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11PixelShader interface (d3d11.h)

Article02/16/2023

A pixel-shader interface manages an executable program (a pixel shader) that controls the pixel-shader stage.

## Inheritance

The **ID3D11PixelShader** interface inherits from the **ID3D11DeviceChild** interface.

## Remarks

The pixel-shader interface has no methods; use HLSL to implement your shader functionality. All shaders in are implemented from a common set of features referred to as the common-shader core..

To create a pixel shader interface, call [ID3D11Device::CreatePixelShader](#). Before using a pixel shader you must bind it to the device by calling [ID3D11DeviceContext::PSSetShader](#).

This interface is defined in D3D11.h.

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11DeviceChild](#)

[Shader Interfaces](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection interface (d3d11shader.h)

Article 08/19/2022

A shader-reflection interface accesses shader information.

## Inheritance

The **ID3D11ShaderReflection** interface inherits from the [IUnknown](#) interface.  
**ID3D11ShaderReflection** also has these types of members:

## Methods

The **ID3D11ShaderReflection** interface has these methods.

<a href="#">ID3D11ShaderReflection::GetBitwiseInstructionCount</a>
Gets the number of bitwise instructions. ( <code>ID3D11ShaderReflection.GetBitwiseInstructionCount</code> )
<a href="#">ID3D11ShaderReflection::GetConstantBufferByIndex</a>
The <code>ID3D11ShaderReflection::GetConstantBufferByIndex</code> ( <code>d3d11shader.h</code> ) method gets a constant buffer by index.
<a href="#">ID3D11ShaderReflection::GetConstantBufferByName</a>
Get a constant buffer by name. ( <code>ID3D11ShaderReflection.GetConstantBufferByName</code> )
<a href="#">ID3D11ShaderReflection::GetConversionInstructionCount</a>
Gets the number of conversion instructions. ( <code>ID3D11ShaderReflection.GetConversionInstructionCount</code> )
<a href="#">ID3D11ShaderReflection::GetDesc</a>
Get a shader description. ( <code>ID3D11ShaderReflection.GetDesc</code> )
<a href="#">ID3D11ShaderReflection::GetGSIInputPrimitive</a>
Gets the geometry-shader input-primitive description. ( <code>ID3D11ShaderReflection.GetGSIInputPrimitive</code> )

<a href="#">ID3D11ShaderReflection::GetInputParameterDesc</a>
Get an input-parameter description for a shader. (ID3D11ShaderReflection.GetInputParameterDesc)
<a href="#">ID3D11ShaderReflection::GetMinFeatureLevel</a>
Gets the minimum feature level. (ID3D11ShaderReflection.GetMinFeatureLevel)
<a href="#">ID3D11ShaderReflection::GetMovcInstructionCount</a>
Gets the number of Movc instructions. (ID3D11ShaderReflection.GetMovcInstructionCount)
<a href="#">ID3D11ShaderReflection::GetMovInstructionCount</a>
Gets the number of Mov instructions. (ID3D11ShaderReflection.GetMovInstructionCount)
<a href="#">ID3D11ShaderReflection::GetNumInterfaceSlots</a>
Gets the number of interface slots in a shader. (ID3D11ShaderReflection.GetNumInterfaceSlots)
<a href="#">ID3D11ShaderReflection::GetOutputParameterDesc</a>
Get an output-parameter description for a shader. (ID3D11ShaderReflection.GetOutputParameterDesc)
<a href="#">ID3D11ShaderReflection::GetPatchConstantParameterDesc</a>
Get a patch-constant parameter description for a shader.
<a href="#">ID3D11ShaderReflection::GetRequiresFlags</a>
Gets a group of flags that indicates the requirements of a shader. (ID3D11ShaderReflection.GetRequiresFlags)
<a href="#">ID3D11ShaderReflection::GetResourceBindingDesc</a>
Get a description of how a resource is bound to a shader. (ID3D11ShaderReflection.GetResourceBindingDesc)
<a href="#">ID3D11ShaderReflection::GetResourceBindingDescByName</a>
Get a description of how a resource is bound to a shader. (ID3D11ShaderReflection.GetResourceBindingDescByName)
<a href="#">ID3D11ShaderReflection::GetThreadGroupSize</a>
Retrieves the sizes, in units of threads, of the X, Y, and Z dimensions of the shader's thread-group grid. (ID3D11ShaderReflection.GetThreadGroupSize)

## [ID3D11ShaderReflection::GetVariableByName](#)

Gets a variable by name. ([ID3D11ShaderReflection.GetVariableByName](#))

## [ID3D11ShaderReflection::IsSampleFrequencyShader](#)

Indicates whether a shader is a sample frequency shader.

([ID3D11ShaderReflection.IsSampleFrequencyShader](#))

## Remarks

An **ID3D11ShaderReflection** interface can be retrieved for a shader by using [D3DReflect](#).

The following code illustrates retrieving a **ID3D11ShaderReflection** from a shader.

```
pd3dDevice->CreatePixelShader( pPixelShader->GetBufferPointer(),
                                 pPixelShader->GetBufferSize(),
                                 g_pPSClassLinkage, &g_pPixelShader );

ID3D11ShaderReflection* pReflector = NULL;
D3DReflect( pPixelShader->GetBufferPointer(), pPixelShader-
>GetBufferSize(),
            IID_ID3D11ShaderReflection, (void**) &pReflector);
```

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shader.h

## See also

[IUnknown](#)

[Shader Interfaces](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetBitwiseInstructionCount method (d3d11shader.h)

Article 02/22/2024

Gets the number of bitwise instructions.

## Syntax

C++

```
UINT GetBitwiseInstructionCount();
```

## Return value

Type: [UINT](#)

The number of bitwise instructions.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetConstantBufferByIndex method (d3d11shader.h)

Article 08/19/2022

Get a constant buffer by index.

## Syntax

C++

```
ID3D11ShaderReflectionConstantBuffer * GetConstantBufferByIndex(  
    [in] UINT Index  
);
```

## Parameters

[in] Index

Type: [UINT](#)

Zero-based index.

## Return value

Type: [ID3D11ShaderReflectionConstantBuffer\\*](#)

A pointer to a constant buffer (see [ID3D11ShaderReflectionConstantBuffer 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	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetConstantBufferByName method (d3d11shader.h)

Article 02/22/2024

Get a constant buffer by name.

## Syntax

C++

```
ID3D11ShaderReflectionConstantBuffer * GetConstantBufferByName(  
    [in] LPCSTR Name  
);
```

## Parameters

[in] Name

Type: [LPCSTR](#)

The constant-buffer name.

## Return value

Type: [ID3D11ShaderReflectionConstantBuffer\\*](#)

A pointer to a constant buffer (see [ID3D11ShaderReflectionConstantBuffer 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

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetConversionInstructionCount method (d3d11shader.h)

Article 02/22/2024

Gets the number of conversion instructions.

## Syntax

C++

```
UINT GetConversionInstructionCount();
```

## Return value

Type: [UINT](#)

Returns the number of conversion instructions.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetDesc method (d3d11shader.h)

Article 02/22/2024

Get a shader description.

## Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D11_SHADER_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_SHADER\\_DESC\\*](#)

A pointer to a shader description. See [D3D11\\_SHADER\\_DESC](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetGSInputPrimitive method (d3d11shader.h)

Article02/22/2024

Gets the geometry-shader input-primitive description.

## Syntax

C++

```
D3D_PRIMITIVE GetGSInputPrimitive();
```

## Return value

Type: [D3D\\_PRIMITIVE](#)

The input-primitive description. See [D3D\\_PRIMITIVE\\_TOPOLOGY](#), [D3D11\\_PRIMITIVE\\_TOPOLOGY](#), or [D3D10\\_PRIMITIVE\\_TOPOLOGY](#).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetInputParameterDesc method (d3d11shader.h)

Article 07/27/2022

Get an input-parameter description for a shader.

## Syntax

C++

```
HRESULT GetInputParameterDesc(  
    [in]  UINT             ParameterIndex,  
    [out] D3D11_SIGNATURE_PARAMETER_DESC *pDesc  
);
```

## Parameters

[in] ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

[out] pDesc

Type: [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC\\*](#)

A pointer to a shader-input-signature description. See [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 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	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetMinFeatureLevel method (d3d11shader.h)

Article 02/22/2024

Gets the minimum feature level.

## Syntax

C++

```
HRESULT GetMinFeatureLevel(  
    D3D_FEATURE_LEVEL *pLevel  
) ;
```

## Parameters

`pLevel`

Type: [out] [D3D\\_FEATURE\\_LEVEL\\*](#)

A pointer to one of the enumerated values in [D3D\\_FEATURE\\_LEVEL](#), which represents the minimum feature level.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetMovInstructionCount method (d3d11shader.h)

Article 02/22/2024

Gets the number of Mov instructions.

## Syntax

C++

```
UINT GetMovInstructionCount();
```

## Return value

Type: [UINT](#)

Returns the number of Mov instructions.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetMovcInstructionCount method (d3d11shader.h)

Article 02/22/2024

Gets the number of Movc instructions.

## Syntax

C++

```
UINT GetMovcInstructionCount();
```

## Return value

Type: [UINT](#)

Returns the number of Movc instructions.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetNumInterfaceSlots method (d3d11shader.h)

Article 02/22/2024

Gets the number of interface slots in a shader.

## Syntax

C++

```
UINT GetNumInterfaceSlots();
```

## Return value

Type: [UINT](#)

The number of interface slots in the shader.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetOutputParameterDesc method (d3d11shader.h)

Article 02/22/2024

Get an output-parameter description for a shader.

## Syntax

C++

```
HRESULT GetOutputParameterDesc(
    [in]  UINT             ParameterIndex,
    [out] D3D11_SIGNATURE_PARAMETER_DESC *pDesc
);
```

## Parameters

[in] ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

[out] pDesc

Type: [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC\\*](#)

A pointer to a shader-output-parameter description. See [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An output-parameter description is also called a shader signature. The shader signature contains information about the output parameters such as the order or parameters,

their data type, and a parameter semantic.

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetPatchConstantParameterDesc method (d3d11shader.h)

Article 02/22/2024

Get a patch-constant parameter description for a shader.

## Syntax

C++

```
HRESULT GetPatchConstantParameterDesc(
    [in]    UINT           ParameterIndex,
    [out]   D3D11_SIGNATURE_PARAMETER_DESC *pDesc
);
```

## Parameters

[in] ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

[out] pDesc

Type: [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC\\*](#)

A pointer to a shader-input-signature description. See [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetRequiresFlags method (d3d11shader.h)

Article02/22/2024

Gets a group of flags that indicates the requirements of a shader.

## Syntax

C++

```
UINT64 GetRequiresFlags();
```

## Return value

Type: **UINT64**

A value that contains a combination of one or more shader requirements flags; each flag specifies a requirement of the shader. A default value of 0 means there are no requirements.

 Expand table

Shader requirement flag	Description
D3D_SHADER_REQUIRES_DOUBLES	Shader requires that the graphics driver and hardware support double data type. For more info, see <a href="#">D3D11_FEATURE_DATA_DOUBLE</a> .
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 <a href="#">Using HLSL minimum precision</a> .
D3D_SHADER_REQUIRES_11_1_DOUBLE_EXTENSIONS	Shader requires that the graphics driver and hardware support extended doubles instructions. For more info, see the <a href="#">ExtendedDoublesShaderInstructions</a> member of <a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS</a> .
D3D_SHADER_REQUIRES_11_1_SHADER_EXTENSIONS	Shader requires that the graphics driver and hardware support the <a href="#">msad4</a> intrinsic function in shaders. For more info, see the

	SAD4ShaderInstructions member of <a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS</a> .
D3D_SHADER_REQUIRES_LEVEL_9_COMPARISON_FILTERING	Shader requires that the graphics driver and hardware support Direct3D 9 shadow support. For more info, see <a href="#">D3D11_FEATURE_DATA_D3D9_SHADOW_SUPPORT</a> .
D3D_SHADER_REQUIRES_TILED_RESOURCES	Shader requires that the graphics driver and hardware support tiled resources. For more info, see <a href="#">GetResourceTiling</a> .

## Remarks

Here is how the D3D11Shader.h header defines the shader requirements flags:

C++

```
#define D3D_SHADER_REQUIRES_DOUBLES          0x00000001
#define D3D_SHADER_REQUIRES_EARLY_DEPTH_STENCIL 0x00000002
#define D3D_SHADER_REQUIRES_UAVS_AT_EVERY_STAGE 0x00000004
#define D3D_SHADER_REQUIRES_64_UAVS             0x00000008
#define D3D_SHADER_REQUIRES_MINIMUM_PRECISION   0x00000010
#define D3D_SHADER_REQUIRES_11_1_DOUBLE_EXTENSIONS 0x00000020
#define D3D_SHADER_REQUIRES_11_1_SHADER_EXTENSIONS 0x00000040
#define D3D_SHADER_REQUIRES_LEVEL_9_COMPARISON_FILTERING 0x00000080
```

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler_47.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetResourceBindingDesc method (d3d11shader.h)

Article 02/22/2024

Get a description of how a resource is bound to a shader.

## Syntax

C++

```
HRESULT GetResourceBindingDesc(  
    [in]  UINT             ResourceIndex,  
    [out] D3D11_SHADER_INPUT_BIND_DESC *pDesc  
>;
```

## Parameters

[in] ResourceIndex

Type: [UINT](#)

A zero-based resource index.

[out] pDesc

Type: [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC\\*](#)

A pointer to an input-binding description. See [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

A shader consists of executable code (the compiled HLSL functions) and a set of resources that supply the shader with input data. [GetResourceBindingDesc](#) gets

information about how one resource in the set is bound as an input to the shader. The *ResourceIndex* parameter specifies the index for the resource.

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetResourceBindingDescByName method (d3d11shader.h)

Article 02/22/2024

Get a description of how a resource is bound to a shader.

## Syntax

C++

```
HRESULT GetResourceBindingDescByName(
    [in]    LPCSTR           Name,
    [out]   D3D11_SHADER_INPUT_BIND_DESC *pDesc
);
```

## Parameters

[in] Name

Type: [LPCSTR](#)

The constant-buffer name of the resource.

[out] pDesc

Type: [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC\\*](#)

A pointer to an input-binding description. See [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

A shader consists of executable code (the compiled HLSL functions) and a set of resources that supply the shader with input data. `GetResourceBindingDescByName` gets information about how one resource in the set is bound as an input to the shader. The *Name* parameter specifies the name of the resource.

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetThreadGroupSize method (d3d11shader.h)

Article 02/22/2024

Retrieves the sizes, in units of threads, of the X, Y, and Z dimensions of the shader's thread-group grid.

## Syntax

C++

```
UINT GetThreadGroupSize(  
    [out, optional] UINT *pSizeX,  
    [out, optional] UINT *pSizeY,  
    [out, optional] UINT *pSizeZ  
) ;
```

## Parameters

[out, optional] pSizeX

Type: **UINT\***

A pointer to the size, in threads, of the x-dimension of the thread-group grid. The maximum size is 1024.

[out, optional] pSizeY

Type: **UINT\***

A pointer to the size, in threads, of the y-dimension of the thread-group grid. The maximum size is 1024.

[out, optional] pSizeZ

Type: **UINT\***

A pointer to the size, in threads, of the z-dimension of the thread-group grid. The maximum size is 64.

## Return value

Type: [UINT](#)

Returns the total size, in threads, of the thread-group grid by calculating the product of the size of each dimension.

#### syntax

```
*pSizeX * *pSizeY * *pSizeZ;
```

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

When a compute shader is written it defines the actions of a single thread group only. If multiple thread groups are required, it is the role of the [ID3D11DeviceContext::Dispatch](#) call to issue multiple thread groups.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::GetVariableByName method (d3d11shader.h)

Article 02/22/2024

Gets a variable by name.

## Syntax

C++

```
ID3D11ShaderReflectionVariable * GetVariableByName(  
    [in] LPCSTR Name  
) ;
```

## Parameters

[in] Name

Type: [LPCSTR](#)

A pointer to a string containing the variable name.

## Return value

Type: [ID3D11ShaderReflectionVariable\\*](#)

Returns a [ID3D11ShaderReflectionVariable Interface](#) interface.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflection::IsSampleFrequencyShader method (d3d11shader.h)

Article 02/22/2024

Indicates whether a shader is a sample frequency shader.

## Syntax

C++

```
BOOL IsSampleFrequencyShader();
```

## Return value

Type: **BOOL**

Returns true if the shader is a sample frequency shader; otherwise returns false.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflection Interface](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionConstantBuffer interface (d3d11shader.h)

Article 02/22/2024

This shader-reflection interface provides access to a constant buffer.

## Methods

The **ID3D11ShaderReflectionConstantBuffer** interface has these methods.

[+] Expand table

<a href="#">ID3D11ShaderReflectionConstantBuffer::GetDesc</a>
Get a constant-buffer description. ( <a href="#">ID3D11ShaderReflectionConstantBuffer.GetDesc</a> )
<a href="#">ID3D11ShaderReflectionConstantBuffer::GetVariableByIndex</a>
The <a href="#">ID3D11ShaderReflectionConstantBuffer::GetVariableByIndex</a> (d3d11shader.h) method gets a shader-reflection variable by index.
<a href="#">ID3D11ShaderReflectionConstantBuffer::GetVariableByName</a>
Get a shader-reflection variable by name. ( <a href="#">ID3D11ShaderReflectionConstantBuffer.GetVariableByName</a> )

## Remarks

To create a constant-buffer interface, call

[ID3D11ShaderReflection::GetConstantBufferByIndex](#) or

[ID3D11ShaderReflection::GetConstantBufferByName](#). This isn't a COM interface, so you don't need to worry about reference counts or releasing the interface when you're done with it.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shader.h

## See also

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionConstantBuffer::GetDesc method (d3d11shader.h)

Article 02/22/2024

Get a constant-buffer description.

## Syntax

C++

```
HRESULT GetDesc(  
    D3D11_SHADER_BUFFER_DESC *pDesc  
);
```

## Parameters

pDesc

Type: [D3D11\\_SHADER\\_BUFFER\\_DESC\\*](#)

A pointer to a [D3D11\\_SHADER\\_BUFFER\\_DESC](#), which represents a shader-buffer description.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionConstantBuffer Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionConstantBuffer::GetVariableByIndex method (d3d11shader.h)

Article 02/22/2024

Get a shader-reflection variable by index.

## Syntax

C++

```
ID3D11ShaderReflectionVariable * GetVariableByIndex(  
    [in] UINT Index  
);
```

## Parameters

[in] Index

Type: [UINT](#)

Zero-based index.

## Return value

Type: [ID3D11ShaderReflectionVariable\\*](#)

A pointer to a shader-reflection variable interface (see [ID3D11ShaderReflectionVariable Interface](#)).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionConstantBuffer Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionConstantBuffer::GetVariableByName method (d3d11shader.h)

Article 02/22/2024

Get a shader-reflection variable by name.

## Syntax

C++

```
ID3D11ShaderReflectionVariable * GetVariableByName(  
    [in] LPCSTR Name  
);
```

## Parameters

[in] Name

Type: [LPCSTR](#)

Variable name.

## Return value

Type: [ID3D11ShaderReflectionVariable\\*](#)

Returns a sentinel object (end of list marker). To determine if GetVariableByName successfully completed, call [ID3D11ShaderReflectionVariable::GetDesc](#) and check the returned **HRESULT**; any return value other than success means that GetVariableByName failed.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionConstantBuffer Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType interface (d3d11shader.h)

Article 08/19/2022

This shader-reflection interface provides access to variable type.

## Methods

The ID3D11ShaderReflectionType interface has these methods.

<a href="#">ID3D11ShaderReflectionType::GetBaseClass</a>
Gets an ID3D11ShaderReflectionType Interface interface containing the variable base class type.
<a href="#">ID3D11ShaderReflectionType::GetDesc</a>
Get the description of a shader-reflection-variable type. (ID3D11ShaderReflectionType.GetDesc)
<a href="#">ID3D11ShaderReflectionType::GetInterfaceByIndex</a>
Get an interface by index.
<a href="#">ID3D11ShaderReflectionType::GetMemberTypeByIndex</a>
The ID3D11ShaderReflectionType::GetMemberTypeByIndex (d3d11shader.h) method gets a shader-reflection-variable type by index.
<a href="#">ID3D11ShaderReflectionType::GetMemberTypeByName</a>
Get a shader-reflection-variable type by name. (ID3D11ShaderReflectionType.GetMemberTypeByName)
<a href="#">ID3D11ShaderReflectionType::GetMemberTypeName</a>
Get a shader-reflection-variable type. (ID3D11ShaderReflectionType.GetMemberTypeName)
<a href="#">ID3D11ShaderReflectionType::GetNumInterfaces</a>
Gets the number of interfaces. (ID3D11ShaderReflectionType.GetNumInterfaces)
<a href="#">ID3D11ShaderReflectionType::GetSubType</a>
Gets the base class of a class. (ID3D11ShaderReflectionType.GetSubType)

### [ID3D11ShaderReflectionType::ImplementsInterface](#)

Indicates whether a class type implements an interface.  
(ID3D11ShaderReflectionType::ImplementsInterface)

### [ID3D11ShaderReflectionType::IsEqual](#)

Indicates whether two ID3D11ShaderReflectionType Interface pointers have the same underlying type.

### [ID3D11ShaderReflectionType::IsOfType](#)

Indicates whether a variable is of the specified type. (ID3D11ShaderReflectionType::IsOfType)

## Remarks

To get a shader-reflection-type interface, call

[ID3D11ShaderReflectionVariable::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

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shader.h

## See also

[Shader Interfaces](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3D11ShaderReflectionType::GetBaseClass method (d3d11shader.h)

Article 02/22/2024

Gets an [ID3D11ShaderReflectionType Interface](#) interface containing the variable base class type.

## Syntax

C++

```
ID3D11ShaderReflectionType * GetBaseClass();
```

## Return value

Type: [ID3D11ShaderReflectionType\\*](#)

Returns A pointer to a [ID3D11ShaderReflectionType Interface](#).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::GetDesc method (d3d11shader.h)

Article 02/22/2024

Get the description of a shader-reflection-variable type.

## Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D11_SHADER_TYPE_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_SHADER\\_TYPE\\_DESC\\*](#)

A pointer to a shader-type description (see [D3D11\\_SHADER\\_TYPE\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::GetInterfaceByIndex method (d3d11shader.h)

Article 02/22/2024

Get an interface by index.

## Syntax

C++

```
ID3D11ShaderReflectionType * GetInterfaceByIndex(  
    [in] UINT uIndex  
) ;
```

## Parameters

[in] uIndex

Type: [UINT](#)

Zero-based index.

## Return value

Type: [ID3D11ShaderReflectionType\\*](#)

A pointer to a [ID3D11ShaderReflectionType](#) interface.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::GetMemberTypeByIndex method (d3d11shader.h)

Article 02/22/2024

Get a shader-reflection-variable type by index.

## Syntax

C++

```
ID3D11ShaderReflectionType * GetMemberTypeByIndex(  
    [in] UINT Index  
);
```

## Parameters

[in] Index

Type: [UINT](#)

Zero-based index.

## Return value

Type: [ID3D11ShaderReflectionType\\*](#)

A pointer to a [ID3D11ShaderReflectionType](#) interface.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::GetMemberTypeByName method (d3d11shader.h)

Article 02/22/2024

Get a shader-reflection-variable type by name.

## Syntax

C++

```
ID3D11ShaderReflectionType * GetMemberTypeByName(  
    [in] LPCSTR Name  
);
```

## Parameters

[in] Name

Type: [LPCSTR](#)

Member name.

## Return value

Type: [ID3D11ShaderReflectionType\\*](#)

A pointer to a [ID3D11ShaderReflectionType](#) interface.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::GetMemberTypeName method (d3d11shader.h)

Article 02/22/2024

Get a shader-reflection-variable type.

## Syntax

C++

```
LPCSTR GetMemberTypeName(  
    [in] UINT Index  
);
```

## Parameters

[in] Index

Type: [UINT](#)

Zero-based index.

## Return value

Type: [LPCSTR](#)

The variable type.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::GetNumInterfaces method (d3d11shader.h)

Article 02/22/2024

Gets the number of interfaces.

## Syntax

C++

```
UINT GetNumInterfaces();
```

## Return value

Type: [UINT](#)

Returns the number of interfaces.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::GetSubType method (d3d11shader.h)

Article 02/22/2024

Gets the base class of a class.

## Syntax

C++

```
ID3D11ShaderReflectionType * GetSubType();
```

## Return value

Type: [ID3D11ShaderReflectionType\\*](#)

Returns a pointer to a [ID3D11ShaderReflectionType Interface](#) containing the base class type. Returns **NULL** if the class does not have a base class.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::ImplementsInterface method (d3d11shader.h)

Article 02/22/2024

Indicates whether a class type implements an interface.

## Syntax

C++

```
HRESULT ImplementsInterface(  
    [in] ID3D11ShaderReflectionType *pBase  
);
```

## Parameters

[in] pBase

Type: [ID3D11ShaderReflectionType\\*](#)

A pointer to a [ID3D11ShaderReflectionType](#) Interface.

## Return value

Type: [HRESULT](#)

Returns S\_OK if the interface is implemented; otherwise return S\_FALSE.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionType::IsEqual method (d3d11shader.h)

Article 02/22/2024

Indicates whether two [ID3D11ShaderReflectionType Interface](#) pointers have the same underlying type.

## Syntax

C++

```
HRESULT IsEqual(
    [in] ID3D11ShaderReflectionType *pType
);
```

## Parameters

[in] pType

Type: [ID3D11ShaderReflectionType\\*](#)

A pointer to a [ID3D11ShaderReflectionType 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 [ID3D11ShaderReflectionType Interface](#) pointers have the same underlying type. For example, if two [ID3D11ShaderReflectionType Interface](#) pointers were retrieved from variables, IsEqual can be used to see if the variables have the same type.

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

# ID3D11ShaderReflectionType::IsOfType method (d3d11shader.h)

Article 02/22/2024

Indicates whether a variable is of the specified type.

## Syntax

C++

```
HRESULT IsOfType(  
    [in] ID3D11ShaderReflectionType *pType  
);
```

## Parameters

[in] pType

Type: [ID3D11ShaderReflectionType\\*](#)

A pointer to a [ID3D11ShaderReflectionType](#) interface.

## Return value

Type: [HRESULT](#)

Returns S\_OK if object being queried is equal to or inherits from the type in the *pType* parameter; otherwise returns S\_FALSE.

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionType Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionVariable interface (d3d11shader.h)

Article02/22/2024

This shader-reflection interface provides access to a variable.

## Methods

The **ID3D11ShaderReflectionVariable** interface has these methods.

[+] Expand table

<a href="#">ID3D11ShaderReflectionVariable::GetBuffer</a>
This method returns the buffer of the current ID3D11ShaderReflectionVariable.
<a href="#">ID3D11ShaderReflectionVariable::GetDesc</a>
Get a shader-variable description. ( <code>ID3D11ShaderReflectionVariable.GetDesc</code> )
<a href="#">ID3D11ShaderReflectionVariable::GetInterfaceSlot</a>
Gets the corresponding interface slot for a variable that represents an interface pointer. ( <code>ID3D11ShaderReflectionVariable.GetInterfaceSlot</code> )
<a href="#">ID3D11ShaderReflectionVariable::GetType</a>
Get a shader-variable type. ( <code>ID3D11ShaderReflectionVariable.GetType</code> )

## Remarks

To get a shader-reflection-variable interface, call a method like [ID3D11ShaderReflection::GetVariableByName](#). This isn't a COM interface, so you don't need to worry about reference counts or releasing the interface when you're done with it.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shader.h

## See also

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionVariable::GetBuffer method (d3d11shader.h)

Article02/22/2024

This method returns the buffer of the current [ID3D11ShaderReflectionVariable](#).

## Syntax

C++

```
ID3D11ShaderReflectionConstantBuffer * GetBuffer();
```

## Return value

Type: [ID3D11ShaderReflectionConstantBuffer\\*](#)

Returns a pointer to the [ID3D11ShaderReflectionConstantBuffer](#) of the present [ID3D11ShaderReflectionVariable](#).

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h

## See also

[ID3D11ShaderReflectionVariable](#)

## Feedback

Was this page helpful?

 Yes

 No



# ID3D11ShaderReflectionVariable::GetDesc method (d3d11shader.h)

Article 02/22/2024

Get a shader-variable description.

## Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D11_SHADER_VARIABLE_DESC *pDesc  
);
```

## Parameters

[out] pDesc

Type: [D3D11\\_SHADER\\_VARIABLE\\_DESC\\*](#)

A pointer to a shader-variable description (see [D3D11\\_SHADER\\_VARIABLE\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

This method can be used to determine if the [ID3D11ShaderReflectionVariable Interface](#) is valid, the method returns [E\\_FAIL](#) when the variable is not valid.

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionVariable Interface](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionVariable::GetInterfaceSlot method (d3d11shader.h)

Article 07/27/2022

Gets the corresponding interface slot for a variable that represents an interface pointer.

## Syntax

C++

```
UINT GetInterfaceSlot(  
    [in] UINT uArrayIndex  
>;
```

## Parameters

[in] uArrayIndex

Type: [UINT](#)

Index of the array element to get the slot number for. For a non-array variable this value will be zero.

## Return value

Type: [UINT](#)

Returns the index of the interface in the interface array.

## Remarks

GetInterfaceSlot gets the corresponding slot in a dynamic linkage array for an interface instance. The returned slot number is used to set an interface instance to a particular class instance. See the HLSL [Interfaces and Classes](#) overview for additional information.

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Examples

## Retrieving and using an interface slot

```
ID3D11ShaderReflectionVariable* pAmbientLightingVar = pReflector->GetVariableByName("g_abstractAmbientLighting");
g_iAmbientLightingOffset = pAmbientLightingVar->GetInterfaceSlot(0);
g_pPSClassLinkage->GetClassInstance( "g_hemiAmbientLight", 0,
&g_pHemiAmbientLightClass );
g_dynamicLinkageArray[g_iAmbientLightingOffset] = g_pHemiAmbientLightClass;
...
pd3dImmediateContext->PSSetShader( g_pPixelShader, g_dynamicLinkageArray,
g_iNumPSInterfaces );
```

## Requirements

Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionVariable Interface](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderReflectionVariable::GetType method (d3d11shader.h)

Article 02/22/2024

Get a shader-variable type.

## Syntax

C++

```
ID3D11ShaderReflectionType * GetType();
```

## Return value

Type: [ID3D11ShaderReflectionType\\*](#)

A pointer to a [ID3D11ShaderReflectionType Interface](#).

## Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler\_xx.dll.

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11shader.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3D11ShaderReflectionVariable Interface](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace interface (d3d11shadertracing.h)

Article 07/22/2021

An **ID3D11ShaderTrace** interface implements methods for obtaining traces of shader executions.

## Inheritance

The **ID3D11ShaderTrace** interface inherits from the [IUnknown](#) interface.

**ID3D11ShaderTrace** also has these types of members:

## Methods

The **ID3D11ShaderTrace** interface has these methods.

<a href="#">ID3D11ShaderTrace::GetInitialRegisterContents</a>
Retrieves the initial contents of the specified input register.
<a href="#">ID3D11ShaderTrace::GetReadRegister</a>
Retrieves information about a register that was read by a step in the trace.
<a href="#">ID3D11ShaderTrace::GetStep</a>
Retrieves information about the specified step in the trace.
<a href="#">ID3D11ShaderTrace::GetTraceStats</a>
Returns statistics about the trace.
<a href="#">ID3D11ShaderTrace::GetWrittenRegister</a>
Retrieves information about a register that was written by a step in the trace.
<a href="#">ID3D11ShaderTrace::PSSelectStamp</a>
Sets the specified pixel-shader stamp.

#### [ID3D11ShaderTrace::ResetTrace](#)

Resets the shader-trace object.

#### [ID3D11ShaderTrace::TraceReady](#)

Specifies that the shader trace recorded and is ready to use.

## Remarks

To retrieve an instance of [ID3D11ShaderTrace](#), call the [ID3D11ShaderTraceFactory::CreateShaderTrace](#) method. To retrieve an instance of [ID3D11ShaderTraceFactory](#), call [IUnknown::QueryInterface](#) on a [ID3D11Device](#) that you created with [D3D11\\_CREATE\\_DEVICE\\_DEBUGGABLE](#). Although shader tracing operates without setting [D3D11\\_CREATE\\_DEVICE\\_DEBUGGABLE](#), we recommend that you create a shader debugging device because some devices (for example, [WARP](#) devices) might make behind-the-scenes shader optimizations that will lead to slightly incorrect shader traces when [D3D11\\_CREATE\\_DEVICE\\_DEBUGGABLE](#) isn't set.

All [ID3D11ShaderTrace](#) methods are thread safe.

All [ID3D11ShaderTrace](#) methods immediately force the reference device to flush rendering commands. Therefore, the most current trace status is always available on the reference device. That is, if you expect a trace to be ready after a draw operation, it will be ready.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows

Header

d3d11shadertracing.h

## See also

[IUnknown](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace::GetInitialRegisterContents method (d3d11shadertracing.h)

Article 02/22/2024

Retrieves the initial contents of the specified input register.

## Syntax

C++

```
HRESULT GetInitialRegisterContents(
    [in] D3D11_TRACE_REGISTER *pRegister,
    [out] D3D11_TRACE_VALUE     *pValue
);
```

## Parameters

[in] pRegister

A pointer to a [D3D11\\_TRACE\\_REGISTER](#) structure that describes the input register to retrieve the initial contents from. You can retrieve valid initial data from only the following input register types. That is, to retrieve valid data, the **RegType** member of [D3D11\\_TRACE\\_REGISTER](#) must be one of the following values:

- [D3D11\\_TRACE\\_INPUT\\_REGISTER](#)
- [D3D11\\_TRACE\\_INPUT\\_PRIMITIVE\\_ID\\_REGISTER](#)
- [D3D11\\_TRACE\\_IMMEDIATE\\_CONSTANT\\_BUFFER](#)

Valid data is indicated by the **ValidMask** member of the [D3D11\\_TRACE\\_VALUE](#) structure that *pValue* points to.

[out] pValue

A pointer to a [D3D11\\_TRACE\\_VALUE](#) structure. **GetInitialRegisterContents** fills the members of this structure with information about the initial contents.

## Return value

**GetInitialRegisterContents** returns:

- **S\_OK** if the method retrieves the initial register contents.
- **E\_FAIL** if a trace is not available.
- **E\_INVALIDARG** if *pRegister* is invalid or **NULL** or if *pValue* is **NULL**.
- Possibly other error codes that are described in [Direct3D 11 Return Codes](#).

## Remarks

You can call **GetInitialRegisterContents** for registers other than the input register types that are specified in the *pRegister* parameter description. However, **GetInitialRegisterContents** sets the **ValidMask** member of the [D3D11\\_TRACE\\_VALUE](#) structure to which *pValue* points to empty (all zeros, 0000), and the register values that the **Bits** member of [D3D11\\_TRACE\\_VALUE](#) specifies are meaningless. The data that **GetInitialRegisterContents** returns is not affected by stepping in a trace; however, the data that is returned is affected by changing the stamp index through a call to [ID3D11ShaderTrace::PSSelectStamp](#).

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[ID3D11ShaderTrace](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace::GetReadRegister method (d3d11shadertracing.h)

Article 02/22/2024

Retrieves information about a register that was read by a step in the trace.

## Syntax

C++

```
HRESULT GetReadRegister(
    [in]  UINT           stepIndex,
    [in]  UINT           readRegisterIndex,
    [out] D3D11_TRACE_REGISTER *pRegister,
    [out] D3D11_TRACE_VALUE   *pValue
);
```

## Parameters

[in] stepIndex

The index of the step within the trace. The range of the index is [0...NumTraceSteps-1], where **NumTraceSteps** is a member of the [D3D11\\_TRACE\\_STATS](#) structure. You can retrieve information in any step order.

[in] readRegisterIndex

The index of the register within the trace step. The range of the index is [0...NumRegistersRead-1], where **NumRegistersRead** is a member of the [D3D11\\_TRACE\\_STEP](#) structure.

[out] pRegister

A pointer to a [D3D11\\_TRACE\\_REGISTER](#) structure. **GetReadRegister** fills the members of this structure with information about the register that was read by the step in the trace.

[out] pValue

A pointer to a [D3D11\\_TRACE\\_VALUE](#) structure. **GetReadRegister** fills the members of this structure with information about the value that was read from the register.

# Return value

GetReadRegister returns:

- **S\_OK** if the method retrieves the register information.
- **E\_FAIL** if a trace is not available or if the trace was not created with the D3D11\_SHADER\_TRACE\_FLAG\_RECORD\_REGISTER\_READS flag.
- **E\_INVALIDARG** if *stepIndex* or *readRegisterIndex* is out of range or if *pRegister* or *pValue* is NULL.
- Possibly other error codes that are described in [Direct3D 11 Return Codes](#).

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[ID3D11ShaderTrace](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace::GetStep method (d3d11shadertracing.h)

Article 10/13/2021

Retrieves information about the specified step in the trace.

## Syntax

C++

```
HRESULT GetStep(
    [in]  UINT          stepIndex,
    [out] D3D11_TRACE_STEP *pTraceStep
);
```

## Parameters

[in] stepIndex

The index of the step within the trace. The range of the index is [0...NumTraceSteps-1], where **NumTraceSteps** is a member of the [D3D11\\_TRACE\\_STATS](#) structure. You can retrieve information about a step in any step order.

[out] pTraceStep

A pointer to a [D3D11\\_TRACE\\_STEP](#) structure. **GetStep** fills the members of this structure with information about the trace step that is specified by the *stepIndex* parameter.

## Return value

**GetStep** returns:

- **S\_OK** if the method retrieves the step information.
- **E\_FAIL** if a trace is not available.
- **E\_INVALIDARG** if *stepIndex* is out of range or if *pTraceStep* is NULL.
- Possibly other error codes that are described in [Direct3D 11 Return Codes](#).

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

# Requirements

<b>Minimum supported client</b>	Windows 8 [desktop apps   UWP apps]
<b>Minimum supported server</b>	Windows Server 2012 [desktop apps   UWP apps]
<b>Target Platform</b>	Windows
<b>Header</b>	d3d11shadertracing.h
<b>DLL</b>	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[ID3D11ShaderTrace](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace::GetTraceStats method (d3d11shadertracing.h)

Article 02/22/2024

Returns statistics about the trace.

## Syntax

C++

```
HRESULT GetTraceStats(  
    [out] D3D11_TRACE_STATS *pTraceStats  
);
```

## Parameters

[out] *pTraceStats*

A pointer to a [D3D11\\_TRACE\\_STATS](#) structure. **GetTraceStats** fills the members of this structure with statistics about the trace.

## Return value

**GetTraceStats** returns:

- S\_OK if statistics about the trace are successfully obtained.
- E\_FAIL if no trace statistics are available yet; [ID3D11ShaderTrace::TraceReady](#) must return S\_OK before **GetTraceStats** can succeed.
- E\_INVALIDARG if *pTraceStats* is NULL.
- Possibly other error codes that are described in [Direct3D 11 Return Codes](#).

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[ID3D11ShaderTrace](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace::GetWrittenRegister method (d3d11shadertracing.h)

Article 10/13/2021

Retrieves information about a register that was written by a step in the trace.

## Syntax

C++

```
HRESULT GetWrittenRegister(
    [in]  UINT             stepIndex,
    [in]  UINT             writtenRegisterIndex,
    [out] D3D11_TRACE_REGISTER *pRegister,
    [out] D3D11_TRACE_VALUE   *pValue
);
```

## Parameters

[in] stepIndex

The index of the step within the trace. The range of the index is [0...NumTraceSteps-1], where **NumTraceSteps** is a member of the [D3D11\\_TRACE\\_STATS](#) structure. You can retrieve information in any step order.

[in] writtenRegisterIndex

The index of the register within the trace step. The range of the index is [0...NumRegistersWritten-1], where **NumRegistersWritten** is a member of the [D3D11\\_TRACE\\_STEP](#) structure.

[out] pRegister

A pointer to a [D3D11\\_TRACE\\_REGISTER](#) structure. **GetWrittenRegister** fills the members of this structure with information about the register that was written by the step in the trace.

[out] pValue

A pointer to a [D3D11\\_TRACE\\_VALUE](#) structure. **GetWrittenRegister** fills the members of this structure with information about the value that was written to the register.

# Return value

`GetWrittenRegister` returns:

- `S_OK` if the method retrieves the register information.
- `E_FAIL` if a trace is not available or if the trace was not created with the `D3D11_SHADER_TRACE_FLAG_RECORD_REGISTER_WRITES` flag.
- `E_INVALIDARG` if `stepIndex` or `writtenRegisterIndex` is out of range or if `pRegister` or `pValue` is `NULL`.
- Possibly other error codes that are described in [Direct3D 11 Return Codes](#).

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[ID3D11ShaderTrace](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace::PSSelectStamp method (d3d11shadertracing.h)

Article 02/22/2024

Sets the specified pixel-shader stamp.

## Syntax

C++

```
HRESULT PSSelectStamp(  
    [in] UINT stampIndex  
);
```

## Parameters

[in] stampIndex

The index of the stamp to select.

## Return value

**PSSelectStamp** returns:

- **S\_OK** if the method set the pixel-shader stamp, and if the primitive covers the pixel and sample for the stamp.
- **S\_FALSE** if the method set the pixel-shader stamp, and if the invocation for the selected stamp falls off the primitive.
- **E\_FAIL** if you called the method for a vertex shader or geometry shader; **PSSelectStamp** is meaningful only for pixel shaders.
- **E\_INVALIDARG** if *stampIndex* is out of range [0..3].
- Possibly other error codes that are described in [Direct3D 11 Return Codes](#).

## Remarks

After you call **PSSelectStamp** to set the pixel-shader stamp, you can call the [ID3D11ShaderTrace::GetInitialRegisterContents](#), [ID3D11ShaderTrace::GetStep](#),

[ID3D11ShaderTrace::GetWrittenRegister](#), and [ID3D11ShaderTrace::GetReadRegister](#) methods to get trace data for that stamp.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[ID3D11ShaderTrace](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace::ResetTrace method (d3d11shadertracing.h)

Article 06/29/2021

Resets the shader-trace object.

## Syntax

C++

```
void ResetTrace();
```

## Return value

None

## Remarks

After you call `ResetTrace`, the `ID3D11ShaderTrace` object behaves as if it had just been created. Thereafter, shader invocations for the trace start from 0 again; calls to `ID3D11ShaderTrace::TraceReady` return `S_FALSE` until the selected shader invocation number is reached, and `TraceReady` records a new trace.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h

DLL

D3D11SDKLayers.dll; D3D11\_1SDKLayers.dll; D3D11\_2SDKLayers.dll

## See also

[ID3D11ShaderTrace](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11ShaderTrace::TraceReady method (d3d11shadertracing.h)

Article 02/22/2024

Specifies that the shader trace recorded and is ready to use.

## Syntax

C++

```
HRESULT TraceReady(
    [out, optional] UINT64 *pTestCount
);
```

## Parameters

[out, optional] pTestCount

An optional pointer to a variable that receives the number of times that a matching invocation for the trace occurred. If not used, set to NULL. For more information about this number, see Remarks.

## Return value

TraceReady returns:

- S\_OK if the trace is ready.
- S\_FALSE if the trace is not ready.
- E\_OUTOFMEMORY if memory ran out while the trace was in the process of recording. You can try to record the trace again by calling [ID3D11ShaderTrace::ResetTrace](#) and then redrawing. If you decide not to record the trace again, release the [ID3D11ShaderTrace](#) interface.
- Possibly other error codes that are described in [Direct3D 11 Return Codes](#).

## Remarks

If a trace is meant to record invocation 3 but only two invocations have happened so far, TraceReady sets the variable to which *pTestCount* points to 2. You can use this value to

understand why a trace is not ready yet. Conversely, the variable to which *pTestCount* points might be larger than the requested invocation count for a trace that is ready. You can use this value to determine the number of invocations that ran past the required trace invocation count. For example, you might not know the number of overdraws that occur on a pixel for a given shader in a draw call. If you can redraw the scene identically, you can set up the traces this next time based on the value that **TraceReady** returned at *pTestCount* on the first pass.

If the shader trace recorded, you can successfully call the [ID3D11ShaderTrace::GetTraceStats](#), [ID3D11ShaderTrace::GetInitialRegisterContents](#), and [ID3D11ShaderTrace::GetStep](#) methods. You can call the [ID3D11ShaderTrace::ResetTrace](#) and [ID3D11ShaderTrace::PSSelectStamp](#) methods regardless of whether the shader trace recorded.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[ID3D11ShaderTrace](#)

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3D11ShaderTraceFactory interface (d3d11shadertracing.h)

Article 02/22/2024

An **ID3D11ShaderTraceFactory** interface implements a method for generating shader trace information objects.

## Inheritance

The **ID3D11ShaderTraceFactory** interface inherits from the [IUnknown](#) interface.  
**ID3D11ShaderTraceFactory** also has these types of members:

## Methods

The **ID3D11ShaderTraceFactory** interface has these methods.

[+] Expand table

<p><a href="#">ID3D11ShaderTraceFactory::CreateShaderTrace</a></p> <p>Creates a shader-trace interface for a shader-trace information object.</p>

## Remarks

These APIs require the Windows Software Development Kit (SDK) for Windows 8.

To retrieve an instance of **ID3D11ShaderTraceFactory**, call [IUnknown::QueryInterface](#) on a [ID3D11Device](#) that you created with [D3D11\\_CREATE\\_DEVICE\\_DEBUGGABLE](#).

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h

## See also

[IUnknown](#)

[Shader Interfaces](#)

---

## Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3D11ShaderTraceFactory::CreateShaderTrace method (d3d11shadertracing.h)

Article 10/13/2021

Creates a shader-trace interface for a shader-trace information object.

## Syntax

C++

```
HRESULT CreateShaderTrace(
    [in] IUnknown                 *pShader,
    [in] D3D11_SHADER_TRACE_DESC *pTraceDesc,
    [out] ID3D11ShaderTrace     **ppShaderTrace
);
```

## Parameters

[in] pShader

A pointer to the interface of the shader to create the shader-trace interface for. For example, *pShader* can be an instance of [ID3D11VertexShader](#), [ID3D11PixelShader](#), and so on.

[in] pTraceDesc

A pointer to a [D3D11\\_SHADER\\_TRACE\\_DESC](#) structure that describes the shader-trace object to create. This parameter cannot be NULL.

[out] ppShaderTrace

A pointer to a variable that receives a pointer to the [ID3D11ShaderTrace](#) interface for the shader-trace object that [CreateShaderTrace](#) creates.

## Return value

[CreateShaderTrace](#) returns:

- [S\\_OK](#) if the method created the shader-trace information object.
- [E\\_FAIL](#) if the reference device, which supports tracing, is not being used.

- **E\_OUTOFMEMORY** if memory is unavailable to complete the operation.
- **E\_INVALIDARG** if any parameter is NULL or invalid.
- Possibly other error codes that are described in [Direct3D 11 Return Codes](#).

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[ID3D11ShaderTraceFactory](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D11VertexShader interface (d3d11.h)

Article02/16/2023

A vertex-shader interface manages an executable program (a vertex shader) that controls the vertex-shader stage.

## Inheritance

The **ID3D11VertexShader** interface inherits from the **ID3D11DeviceChild** interface.

## Remarks

The vertex-shader interface has no methods; use HLSL to implement your shader functionality. All shaders are implemented from a common set of features referred to as the common-shader core..

To create a vertex shader interface, call [ID3D11Device::CreateVertexShader](#). Before using a vertex shader you must bind it to the device by calling [ID3D11DeviceContext::VSSetShader](#).

This interface is defined in D3D11.h.

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

## See also

[ID3D11DeviceChild](#)

[Shader Interfaces](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Shader Functions (Direct3D 11 Graphics)

Article • 12/10/2020

This section contains information about the shader functions.

## In this section

Topic	Description
<a href="#">D3DDisassemble11Trace</a>	Disassembles a section of compiled Microsoft High Level Shader Language (HLSL) code that is specified by shader trace steps.

## Related topics

[Shader Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DDisassemble11Trace function (d3d11shadertracing.h)

Article 10/13/2021

Disassembles a section of compiled Microsoft High Level Shader Language (HLSL) code that is specified by shader trace steps.

## Syntax

C++

```
HRESULT D3DDisassemble11Trace(
    [in]    LPCVOID           pSrcData,
    [in]    SIZE_T            SrcDataSize,
    [in]    ID3D11ShaderTrace *pTrace,
    [in]    UINT               StartStep,
    [in]    UINT               NumSteps,
    [in]    UINT               Flags,
    [out]   ID3D10Blob        **ppDisassembly
);
```

## Parameters

[in] **pSrcData**

Type: **LPCVOID**

A pointer to compiled shader data.

[in] **SrcDataSize**

Type: **SIZE\_T**

The size, in bytes, of the block of memory that pSrcData points to.

[in] **pTrace**

Type: **ID3D11ShaderTrace\***

A pointer to the ID3D11ShaderTrace interface for the shader trace information object.

[in] **StartStep**

Type: **UINT**

The number of the step in the trace from which D3DDisassemble11Trace starts the disassembly.

[in] NumSteps

Type: **UINT**

The number of trace steps to disassemble.

[in] Flags

Type: **UINT**

A combination of zero or more of the following flags that are combined by using a bitwise OR operation. The resulting value specifies how D3DDisassemble11Trace disassembles the compiled shader data.

 Expand table

Flag	Description
D3D_DISASM_ENABLE_COLOR_CODE (0x01)	Enable the output of color codes.
D3D_DISASM_ENABLE_DEFAULT_VALUE_PRINTS (0x02)	Enable the output of default values.
D3D_DISASM_ENABLE_INSTRUCTION_NUMBERING (0x04)	Enable instruction numbering.
D3D_DISASM_ENABLE_INSTRUCTION_CYCLE (0x08)	No effect.
D3D_DISASM_DISABLE_DEBUG_INFO (0x10)	Disable the output of debug information.
D3D_DISASM_ENABLE_INSTRUCTION_OFFSET (0x20)	Enable the output of instruction offsets.
D3D_DISASM_INSTRUCTION_ONLY (0x40)	Enable the output of the instruction cycle per step in D3DDisassemble11Trace. This flag is similar to the D3D_DISASM_ENABLE_INSTRUCTION_NUMBERING and D3D_DISASM_ENABLE_INSTRUCTION_OFFSET flags. This flag has no effect in the D3DDisassembleRegion function. Cycle information comes from the trace; therefore, cycle information is available only in the trace disassembly.

[out] ppDisassembly

Type: **ID3D10Blob\*\***

A pointer to a buffer that receives the ID3DBlob interface that accesses the disassembled HLSL code.

## Return value

Type: **HRESULT**

This method returns an HRESULT error code.

## Remarks

D3DDisassemble11Trace walks the steps of a shader trace and outputs appropriate disassembly for each step that is based on the step's instruction index. The disassembly is annotated with register-value information from the trace. The behavior of D3DDisassemble11Trace differs from D3DDisassemble in that instead of the static disassembly of a compiled shader that D3DDisassemble performs, D3DDisassemble11Trace provides an execution trace that is based on the shader trace information.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll; D3D11_1SDKLayers.dll; D3D11_2SDKLayers.dll

## See also

[Shader Functions](#)

# Shader Structures (Direct3D 11 Graphics)

Article • 12/10/2020

Structures are used to create and use shaders.

## In this section

Topic	Description
<a href="#">D3D11_CLASS_INSTANCE_DESC</a>	Describes an HLSL class instance.
<a href="#">D3D11_COMPUTE_SHADER_TRACE_DESC</a>	Describes an instance of a compute shader to trace.
<a href="#">D3D11_DOMAIN_SHADER_TRACE_DESC</a>	Describes an instance of a domain shader to trace.
<a href="#">D3D11_FUNCTION_DESC</a>	Describes a function.
<a href="#">D3D11_GEOMETRY_SHADER_TRACE_DESC</a>	Describes an instance of a geometry shader to trace.
<a href="#">D3D11_HULL_SHADER_TRACE_DESC</a>	Describes an instance of a hull shader to trace.
<a href="#">D3D11_LIBRARY_DESC</a>	Describes a library.
<a href="#">D3D11_PARAMETER_DESC</a>	Describes a function parameter.
<a href="#">D3D11_PIXEL_SHADER_TRACE_DESC</a>	Describes an instance of a pixel shader to trace.
<a href="#">D3D11_SHADER_BUFFER_DESC</a>	Describes a shader constant-buffer.
<a href="#">D3D11_SHADER_DESC</a>	Describes a shader.
<a href="#">D3D11_SHADER_INPUT_BIND_DESC</a>	Describes how a shader resource is bound to a shader input.
<a href="#">D3D11_SHADER_TRACE_DESC</a>	Describes a shader-trace object.
<a href="#">D3D11_SHADER_TYPE_DESC</a>	Describes a shader-variable type.
<a href="#">D3D11_SHADER_VARIABLE_DESC</a>	Describes a shader variable.
<a href="#">D3D11_SIGNATURE_PARAMETER_DESC</a>	Describes a shader signature.
<a href="#">D3D11_TRACE_REGISTER</a>	Describes a trace register.
<a href="#">D3D11_TRACE_STATS</a>	Specifies statistics about a trace.
<a href="#">D3D11_TRACE_STEP</a>	Describes a trace step, which is an instruction.
<a href="#">D3D11_TRACE_VALUE</a>	Describes a trace value.

Topic	Description
<a href="#">D3D11_VERTEX_SHADER_TRACE_DESC</a>	Describes an instance of a vertex shader to trace.

## Related topics

[Shader Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D11\_CLASS\_INSTANCE\_DESC structure (d3d11.h)

Article 08/03/2021

Describes an HLSL class instance.

## Syntax

C++

```
typedef struct D3D11_CLASS_INSTANCE_DESC {
    UINT InstanceId;
    UINT InstanceIndex;
    UINT TypeId;
    UINT ConstantBuffer;
    UINT BaseConstantBufferOffset;
    UINT BaseTexture;
    UINT BaseSampler;
    BOOL Created;
} D3D11_CLASS_INSTANCE_DESC;
```

## Members

`InstanceId`

Type: [UINT](#)

The instance ID of an HLSL class; the default value is 0.

`InstanceIndex`

Type: [UINT](#)

The instance index of an HLSL class; the default value is 0.

`TypeId`

Type: [UINT](#)

The type ID of an HLSL class; the default value is 0.

`ConstantBuffer`

Type: [UINT](#)

Describes the constant buffer associated with an HLSL class; the default value is 0.

**BaseConstantBufferOffset**

Type: **UINT**

The base constant buffer offset associated with an HLSL class; the default value is 0.

**BaseTexture**

Type: **UINT**

The base texture associated with an HLSL class; the default value is 127.

**BaseSampler**

Type: **UINT**

The base sampler associated with an HLSL class; the default value is 15.

**Created**

Type: **BOOL**

True if the class was created; the default value is false.

## Remarks

The D3D11\_CLASS\_INSTANCE\_DESC structure is returned by the [ID3D11ClassInstance::GetDesc](#) method.

The members of this structure except **InstanceIndex** are valid (non default values) if they describe a class instance acquired using [ID3D11ClassLinkage::CreateClassInstance](#). The **InstanceIndex** member is only valid when the class instance is acquired using [ID3D11ClassLinkage::GetClassInstance](#).

## Requirements

Header	d3d11.h
--------	---------

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D11\_COMPUTE\_SHADER\_TRACE\_DESC structure (d3d11shadertracing.h)

Article 02/22/2024

Describes an instance of a compute shader to trace.

## Syntax

C++

```
typedef struct D3D11_COMPUTE_SHADER_TRACE_DESC {
    UINT64 Invocation;
    UINT    ThreadIDInGroup[3];
    UINT    ThreadGroupID[3];
} D3D11_COMPUTE_SHADER_TRACE_DESC;
```

## Members

Invocation

The invocation number of the instance of the compute shader.

ThreadIDInGroup[3]

The [SV\\_GroupThreadID](#) to trace. This value identifies indexes of individual threads within a thread group that a compute shader executes in.

ThreadGroupID[3]

The [SV\\_GroupID](#) to trace. This value identifies indexes of a thread group that the compute shader executes in.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_DOMAIN\_SHADER\_TRACE\_DESC structure (d3d11shadertracing.h)

Article02/22/2024

Describes an instance of a domain shader to trace.

## Syntax

C++

```
typedef struct D3D11_DOMAIN_SHADER_TRACE_DESC {
    UINT64 Invocation;
} D3D11_DOMAIN_SHADER_TRACE_DESC;
```

## Members

### Invocation

The invocation number of the instance of the domain shader.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_FUNCTION\_DESC structure (d3d11shader.h)

Article 07/27/2022

Describes a function.

## Syntax

C++

```
typedef struct _D3D11_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;
} D3D11_FUNCTION_DESC;
```

# Members

Version

Type: **UINT**

The shader version.

Creator

Type: **LPCSTR**

The name of the originator of the function.

Flags

Type: **UINT**

A combination of [D3DCOMPILE Constants](#) that are combined by using a bitwise OR operation. The resulting value specifies shader compilation and parsing.

ConstantBuffers

Type: **UINT**

The number of constant buffers for the function.

BoundResources

Type: **UINT**

The number of bound resources for the function.

InstructionCount

Type: **UINT**

The number of emitted instructions for the function.

TempRegisterCount

Type: **UINT**

The number of temporary registers used by the function.

TempArrayCount

Type: **UINT**

The number of temporary arrays used by the function.

`DefCount`

Type: **UINT**

The number of constant defines for the function.

`DclCount`

Type: **UINT**

The number of declarations (input + output) for the function.

`TextureNormalInstructions`

Type: **UINT**

The number of non-categorized texture instructions for the function.

`TextureLoadInstructions`

Type: **UINT**

The number of texture load instructions for the function.

`TextureCompInstructions`

Type: **UINT**

The number of texture comparison instructions for the function.

`TextureBiasInstructions`

Type: **UINT**

The number of texture bias instructions for the function.

`TextureGradientInstructions`

Type: **UINT**

The number of texture gradient instructions for the function.

`FloatingInstructionCount`

Type: **UINT**

The number of floating point arithmetic instructions used by the function.

`IntInstructionCount`

Type: **UINT**

The number of signed integer arithmetic instructions used by the function.

`UintInstructionCount`

Type: **UINT**

The number of unsigned integer arithmetic instructions used by the function.

`StaticFlowControlCount`

Type: **UINT**

The number of static flow control instructions used by the function.

`DynamicFlowControlCount`

Type: **UINT**

The number of dynamic flow control instructions used by the function.

`MacroInstructionCount`

Type: **UINT**

The number of macro instructions used by the function.

`ArrayInstructionCount`

Type: **UINT**

The number of array instructions used by the function.

`MovInstructionCount`

Type: **UINT**

The number of mov instructions used by the function.

`MovcInstructionCount`

Type: **UINT**

The number of movc instructions used by the function.

`ConversionInstructionCount`

Type: **UINT**

The number of type conversion instructions used by the function.

`BitwiseInstructionCount`

Type: **UINT**

The number of bitwise arithmetic instructions used by the function.

`MinFeatureLevel`

Type: **D3D\_FEATURE\_LEVEL**

A **D3D\_FEATURE\_LEVEL**-typed value that specifies the minimum Direct3D feature level target of the function byte code.

`RequiredFeatureFlags`

Type: **UINT64**

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 [ID3D11ShaderReflection::GetRequiresFlags](#).

`Name`

Type: **LPCSTR**

The name of the function.

`FunctionParameterCount`

Type: **INT**

The number of logical parameters in the function signature, not including the return value.

`HasReturn`

Type: **BOOL**

Indicates whether the function returns a value. **TRUE** indicates it returns a value; otherwise, **FALSE** (it is a subroutine).

`Has10Level9VertexShader`

Type: **BOOL**

Indicates whether there is a Direct3D 10Level9 vertex shader blob. **TRUE** indicates there is a 10Level9 vertex shader blob; otherwise, **FALSE**.

Has10Level9PixelShader

Type: **BOOL**

Indicates whether there is a Direct3D 10Level9 pixel shader blob. **TRUE** indicates there is a 10Level9 pixel shader blob; otherwise, **FALSE**.

## Requirements

Header	d3d11shader.h
--------	---------------

## See also

[ID3D11FunctionReflection::GetDesc](#)

[Shader Structures](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# D3D11\_GEOMETRY\_SHADER\_TRACE\_DESC structure (d3d11shadertracing.h)

Article 02/22/2024

Describes an instance of a geometry shader to trace.

## Syntax

C++

```
typedef struct D3D11_GEOMETRY_SHADER_TRACE_DESC {
    UINT64 Invocation;
} D3D11_GEOMETRY_SHADER_TRACE_DESC;
```

## Members

### Invocation

The invocation number of the instance of the geometry shader.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_HULL\_SHADER\_TRACE\_DESC structure (d3d11shadertracing.h)

Article 02/22/2024

Describes an instance of a hull shader to trace.

## Syntax

C++

```
typedef struct D3D11_HULL_SHADER_TRACE_DESC {
    UINT64 Invocation;
} D3D11_HULL_SHADER_TRACE_DESC;
```

## Members

### Invocation

The invocation number of the instance of the hull shader.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_LIBRARY\_DESC structure (d3d11shader.h)

Article 02/22/2024

Describes a library.

## Syntax

C++

```
typedef struct _D3D11_LIBRARY_DESC {
    LPCSTR Creator;
    UINT    Flags;
    UINT    FunctionCount;
} D3D11_LIBRARY_DESC;
```

## Members

Creator

Type: [LPCSTR](#)

The name of the originator of the library.

Flags

Type: [UINT](#)

A combination of [D3DCOMPILE Constants](#) that are combined by using a bitwise OR operation. The resulting value specifies how the compiler compiles.

FunctionCount

Type: [UINT](#)

The number of functions exported from the library.

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11shader.h

## See also

[ID3D11LibraryReflection::GetDesc](#)

[Shader Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_PARAMETER\_DESC structure (d3d11shader.h)

Article 02/22/2024

Describes a function parameter.

## Syntax

C++

```
typedef struct _D3D11_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;
} D3D11_PARAMETER_DESC;
```

## Members

Name

Type: [LPCSTR](#)

The name of the function parameter.

SemanticName

Type: [LPCSTR](#)

The HLSL [semantic](#) that is associated with this function parameter. This name includes the index, for example, SV\_Target[n].

Type

Type: [D3D\\_SHADER\\_VARIABLE\\_TYPE](#)

A [D3D\\_SHADER\\_VARIABLE\\_TYPE](#)-typed value that identifies the variable type for the parameter.

Class

Type: [D3D\\_SHADER\\_VARIABLE\\_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

Type: [UINT](#)

The number of rows for a matrix parameter.

Columns

Type: [UINT](#)

The number of columns for a matrix parameter.

InterpolationMode

Type: [D3D\\_INTERPOLATION\\_MODE](#)

A [D3D\\_INTERPOLATION\\_MODE](#)-typed value that identifies the interpolation mode for the parameter.

Flags

Type: [D3D\\_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

Type: [UINT](#)

The first input register for this parameter.

FirstInComponent

Type: [UINT](#)

The first input register component for this parameter.

`FirstOutRegister`

Type: [UINT](#)

The first output register for this parameter.

`FirstOutComponent`

Type: [UINT](#)

The first output register component for this parameter.

## Remarks

Get a function-parameter description by calling [ID3D11FunctionParameterReflection::GetDesc](#).

## Requirements

[Expand table](#)

Requirement	Value
Header	d3d11shader.h

## See also

[ID3D11FunctionParameterReflection::GetDesc](#)

[Shader Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_PIXEL\_SHADER\_TRACE\_DESC structure (d3d11shadertracing.h)

Article 06/24/2021

Describes an instance of a pixel shader to trace.

## Syntax

C++

```
typedef struct D3D11_PIXEL_SHADER_TRACE_DESC {
    UINT64 Invocation;
    INT     X;
    INT     Y;
    UINT64 SampleMask;
} D3D11_PIXEL_SHADER_TRACE_DESC;
```

## Members

`Invocation`

The invocation number of the instance of the pixel shader.

`X`

The x-coordinate of the pixel.

`Y`

The y-coordinate of the pixel.

`SampleMask`

A value that describes a mask of pixel samples to trace. If this value specifies any of the masked samples, the trace is activated. The least significant bit (LSB) is sample 0. The non-multisample antialiasing (MSAA) counts as a sample count of 1; therefore, the LSB of `SampleMask` should be set. If set to zero, the pixel is not traced. However, pixel traces can still be enabled on an invocation basis.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D11\_SHADER\_BUFFER\_DESC structure (d3d11shader.h)

Article 02/22/2024

Describes a shader constant-buffer.

## Syntax

C++

```
typedef struct _D3D11_SHADER_BUFFER_DESC {
    LPCSTR          Name;
    D3D_CBUFFER_TYPE Type;
    UINT            Variables;
    UINT            Size;
    UINT            uFlags;
} D3D11_SHADER_BUFFER_DESC;
```

## Members

Name

Type: [LPCSTR](#)

The name of the buffer.

Type

Type: [D3D\\_CBUFFER\\_TYPE](#)

A [D3D\\_CBUFFER\\_TYPE](#)-typed value that indicates the intended use of the constant data.

Variables

Type: [UINT](#)

The number of unique variables.

Size

Type: [UINT](#)

Buffer size (in bytes).

## uFlags

Type: [UINT](#)

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 [ID3D11ShaderReflectionConstantBuffer::GetDesc](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3d11shader.h

## See also

[Shader Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_SHADER\_DESC structure (d3d11shader.h)

Article07/27/2022

Describes a shader.

# Syntax

C++

```
typedef struct _D3D11_SHADER_DESC {  
    UINT Version;  
    LPCSTR Creator;  
    Flags;  
    ConstantBuffers;  
    BoundResources;  
    InputParameters;  
    OutputParameters;  
    InstructionCount;  
    TempRegisterCount;  
    TempArrayCount;  
    DefCount;  
    DclCount;  
    TextureNormalInstructions;  
    TextureLoadInstructions;  
    TextureCompInstructions;  
    TextureBiasInstructions;  
    TextureGradientInstructions;  
    FloatInstructionCount;  
    IntInstructionCount;  
    UintInstructionCount;  
    StaticFlowControlCount;  
    DynamicFlowControlCount;  
    MacroInstructionCount;  
    ArrayInstructionCount;  
    CutInstructionCount;  
    EmitInstructionCount;  
    GSOutputTopology;  
    GSMaxOutputVertexCount;  
    InputPrimitive;  
    PatchConstantParameters;  
    cGSInstanceCount;  
    cControlPoints;  
    D3D_PRIMITIVE_TOPOLOGY HSOutputPrimitive;  
    D3D_PRIMITIVE InputPrimitive;  
    D3D_TESSELLATOR_PARTITIONING HSPartitioning;  
    D3D_TESSELLATOR_DOMAIN TessellatorDomain;  
    D3D_TESSELLATOR_OUTPUT_PRIMITIVE cBarrierInstructions;  
    D3D_TESSELLATOR_DOMAIN cInterlockedInstructions;
```

```
    UINT                                     cTextureStoreInstructions;
} D3D11_SHADER_DESC;
```

# Members

## Version

Type: **UINT**

Shader version.

## Creator

Type: **LPCSTR**

The name of the originator of the shader.

## Flags

Type: **UINT**

Shader compilation/parse flags.

## ConstantBuffers

Type: **UINT**

The number of shader-constant buffers.

## BoundResources

Type: **UINT**

The number of resource (textures and buffers) bound to a shader.

## InputParameters

Type: **UINT**

The number of parameters in the input signature.

## OutputParameters

Type: **UINT**

The number of parameters in the output signature.

`InstructionCount`

Type: **UINT**

The number of intermediate-language instructions in the compiled shader.

`TempRegisterCount`

Type: **UINT**

The number of temporary registers in the compiled shader.

`TempArrayCount`

Type: **UINT**

Number of temporary arrays used.

`DefCount`

Type: **UINT**

Number of constant defines.

`DclCount`

Type: **UINT**

Number of declarations (input + output).

`TextureNormalInstructions`

Type: **UINT**

Number of non-categorized texture instructions.

`TextureLoadInstructions`

Type: **UINT**

Number of texture load instructions

`TextureCompInstructions`

Type: **UINT**

Number of texture comparison instructions

`TextureBiasInstructions`

Type: **UINT**

Number of texture bias instructions

`TextureGradientInstructions`

Type: **UINT**

Number of texture gradient instructions.

`FloatInstructionCount`

Type: **UINT**

Number of floating point arithmetic instructions used.

`IntInstructionCount`

Type: **UINT**

Number of signed integer arithmetic instructions used.

`UintInstructionCount`

Type: **UINT**

Number of unsigned integer arithmetic instructions used.

`StaticFlowControlCount`

Type: **UINT**

Number of static flow control instructions used.

`DynamicFlowControlCount`

Type: **UINT**

Number of dynamic flow control instructions used.

`MacroInstructionCount`

Type: **UINT**

Number of macro instructions used.

`ArrayInstructionCount`

Type: **UINT**

Number of array instructions used.

`CutInstructionCount`

Type: [UINT](#)

Number of cut instructions used.

`EmitInstructionCount`

Type: [UINT](#)

Number of emit instructions used.

`GSOutputTopology`

Type: [D3D\\_PRIMITIVE\\_TOPOLOGY](#)

The [D3D\\_PRIMITIVE\\_TOPOLOGY](#)-typed value that represents the geometry shader output topology.

`GSMaxOutputVertexCount`

Type: [UINT](#)

Geometry shader maximum output vertex count.

`InputPrimitive`

Type: [D3D\\_PRIMITIVE](#)

The [D3D\\_PRIMITIVE](#)-typed value that represents the input primitive for a geometry shader or hull shader.

`PatchConstantParameters`

Type: [UINT](#)

Number of parameters in the patch-constant signature.

`cGSInstanceCount`

Type: [UINT](#)

Number of geometry shader instances.

`cControlPoints`

Type: **UINT**

Number of control points in the hull shader and domain shader.

`HSOutputPrimitive`

Type: **D3D\_TESSELLATOR\_OUTPUT\_PRIMITIVE**

The **D3D\_TESSELLATOR\_OUTPUT\_PRIMITIVE**-typed value that represents the tessellator output-primitive type.

`HSPartitioning`

Type: **D3D\_TESSELLATOR\_PARTITIONING**

The **D3D\_TESSELLATOR\_PARTITIONING**-typed value that represents the tessellator partitioning mode.

`TessellatorDomain`

Type: **D3D\_TESSELLATOR\_DOMAIN**

The **D3D\_TESSELLATOR\_DOMAIN**-typed value that represents the tessellator domain.

`cBarrierInstructions`

Type: **UINT**

Number of barrier instructions in a compute shader.

`cInterlockedInstructions`

Type: **UINT**

Number of interlocked instructions in a compute shader.

`cTextureStoreInstructions`

Type: **UINT**

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. Get a shader description by calling [ID3D11ShaderReflection::GetDesc](#).

# Requirements

Header	d3d11shader.h
--------	---------------

## See also

[Shader Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D11\_SHADER\_INPUT\_BIND\_DESC structure (d3d11shader.h)

Article 02/22/2024

Describes how a shader resource is bound to a shader input.

## Syntax

C++

```
typedef struct _D3D11_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;
} D3D11_SHADER_INPUT_BIND_DESC;
```

## Members

Name

Type: [LPCSTR](#)

Name of the shader resource.

Type

Type: [D3D\\_SHADER\\_INPUT\\_TYPE](#)

A [D3D\\_SHADER\\_INPUT\\_TYPE](#)-typed value that identifies the type of data in the resource.

BindPoint

Type: [UINT](#)

Starting bind point.

BindCount

Type: [UINT](#)

Number of contiguous bind points for arrays.

`uFlags`

Type: [UINT](#)

A combination of [D3D\\_SHADER\\_INPUT\\_FLAGS](#)-typed values for shader input-parameter options.

`ReturnType`

Type: [D3D\\_RESOURCE\\_RETURN\\_TYPE](#)

If the input is a texture, the [D3D\\_RESOURCE\\_RETURN\\_TYPE](#)-typed value that identifies the return type.

`Dimension`

Type: [D3D\\_SRV\\_DIMENSION](#)

A [D3D\\_SRV\\_DIMENSION](#)-typed value that identifies the dimensions of the bound resource.

`NumSamples`

Type: [UINT](#)

The number of samples for a multisampled texture; when a texture isn't multisampled, the value is set to -1 (0xFFFFFFFF).

## Remarks

Get a shader-input-signature description by calling [ID3D11ShaderReflection::GetResourceBindingDesc](#) or [ID3D11ShaderReflection::GetResourceBindingDescByName](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11shader.h

## See also

[Shader Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_SHADER\_TRACE\_DESC structure (d3d11shadertracing.h)

Article 02/22/2024

Describes a shader-trace object.

## Syntax

C++

```
typedef struct D3D11_SHADER_TRACE_DESC {
    D3D11_SHADER_TYPE Type;
    UINT              Flags;
    union {
        D3D11_VERTEX_SHADER_TRACE_DESC  VertexShaderTraceDesc;
        D3D11_HULL_SHADER_TRACE_DESC   HullShaderTraceDesc;
        D3D11_DOMAIN_SHADER_TRACE_DESC DomainShaderTraceDesc;
        D3D11_GEOMETRY_SHADER_TRACE_DESC GeometryShaderTraceDesc;
        D3D11_PIXEL_SHADER_TRACE_DESC  PixelShaderTraceDesc;
        D3D11_COMPUTE_SHADER_TRACE_DESC ComputeShaderTraceDesc;
    };
} D3D11_SHADER_TRACE_DESC;
```

## Members

### Type

A [D3D11\\_SHADER\\_TYPE](#)-typed value that identifies the type of shader that the shader-trace object describes. This member also determines which shader-trace type to use in the following union.

### Flags

A combination of the following flags that are combined by using a bitwise **OR** operation. The resulting value specifies how [ID3D11ShaderTraceFactory::CreateShaderTrace](#) creates the shader-trace object.

[+] Expand table

Flag	Description
D3D11_SHADER_TRACE_FLAG_RECORD_REGISTER_WRITES	The shader trace object records

(0x1)	register-writes.
D3D11_SHADER_TRACE_FLAG_RECORD_REGISTER_READS (0x2)	The shader trace object records register-reads.

#### VertexShaderTraceDesc

A [D3D11\\_VERTEX\\_SHADER\\_TRACE\\_DESC](#) structure that describes an instance of a vertex shader to trace.

#### HullShaderTraceDesc

A [D3D11\\_HULL\\_SHADER\\_TRACE\\_DESC](#) structure that describes an instance of a hull shader to trace.

#### DomainShaderTraceDesc

A [D3D11\\_DOMAIN\\_SHADER\\_TRACE\\_DESC](#) structure that describes an instance of a domain shader to trace.

#### GeometryShaderTraceDesc

A [D3D11\\_GEOMETRY\\_SHADER\\_TRACE\\_DESC](#) structure that describes an instance of a geometry shader to trace.

#### PixelShaderTraceDesc

A [D3D11\\_PIXEL\\_SHADER\\_TRACE\\_DESC](#) structure that describes an instance of a pixel shader to trace.

#### ComputeShaderTraceDesc

A [D3D11\\_COMPUTE\\_SHADER\\_TRACE\\_DESC](#) structure that describes an instance of a compute shader to trace.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[ID3D11ShaderTraceFactory::CreateShaderTrace](#)

[Shader Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_SHADER\_TYPE\_DESC structure (d3d11shader.h)

Article 02/22/2024

Describes a shader-variable type.

## Syntax

C++

```
typedef struct _D3D11_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;
} D3D11_SHADER_TYPE_DESC;
```

## Members

Class

Type: [D3D\\_SHADER\\_VARIABLE\\_CLASS](#)

A [D3D\\_SHADER\\_VARIABLE\\_CLASS](#)-typed value that identifies the variable class as one of scalar, vector, matrix, object, and so on.

Type

Type: [D3D\\_SHADER\\_VARIABLE\\_TYPE](#)

A [D3D\\_SHADER\\_VARIABLE\\_TYPE](#)-typed value that identifies the variable type.

Rows

Type: [UINT](#)

Number of rows in a matrix. Otherwise a numeric type returns 1, any other type returns 0.

## Columns

Type: [UINT](#)

Number of columns in a matrix. Otherwise a numeric type returns 1, any other type returns 0.

## Elements

Type: [UINT](#)

Number of elements in an array; otherwise 0.

## Members

Type: [UINT](#)

Number of members in the structure; otherwise 0.

## Offset

Type: [UINT](#)

Offset, in bytes, between the start of the parent structure and this variable. Can be 0 if not a structure member.

## Name

Type: [LPCSTR](#)

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 [ID3D11ShaderReflectionType::GetDesc](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d11shader.h

## See also

[Shader Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_SHADER\_VARIABLE\_DESC structure (d3d11shader.h)

Article 02/22/2024

Describes a shader variable.

## Syntax

C++

```
typedef struct _D3D11_SHADER_VARIABLE_DESC {
    LPCSTR Name;
    UINT StartOffset;
    UINT Size;
    UINT uFlags;
    LPVOID DefaultValue;
    UINT StartTexture;
    UINT TextureSize;
    UINT StartSampler;
    UINT SamplerSize;
} D3D11_SHADER_VARIABLE_DESC;
```

## Members

Name

Type: [LPCSTR](#)

The variable name.

StartOffset

Type: [UINT](#)

Offset from the start of the parent structure to the beginning of the variable.

Size

Type: [UINT](#)

Size of the variable (in bytes).

uFlags

Type: **UINT**

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**

Type: **LPVOID**

The default value for initializing the variable.

**StartTexture**

Type: **UINT**

Offset from the start of the variable to the beginning of the texture.

**TextureSize**

Type: **UINT**

The size of the texture, in bytes.

**StartSampler**

Type: **UINT**

Offset from the start of the variable to the beginning of the sampler.

**SamplerSize**

Type: **UINT**

The size of the sampler, in bytes.

## Remarks

Get a shader-variable description using reflection by calling [ID3D11ShaderReflectionVariable::GetDesc](#).

As of the June 2010 update, **DefaultValue** emits default values for reflection.

## Requirements

[+] Expand table

Requirement	Value
Header	d3d11shader.h

## See also

[Shader Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# D3D11\_SIGNATURE\_PARAMETER\_DESC structure (d3d11shader.h)

Article 07/27/2022

Describes a shader signature.

## Syntax

C++

```
typedef struct _D3D11_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;
} D3D11_SIGNATURE_PARAMETER_DESC;
```

## Members

SemanticName

Type: [LPCSTR](#)

A per-parameter string that identifies how the data will be used. For more info, see [Semantics](#).

SemanticIndex

Type: [UINT](#)

Semantic index that modifies the semantic. Used to differentiate different parameters that use the same semantic.

Register

Type: [UINT](#)

The register that will contain this variable's data.

## SystemValueType

Type: [D3D\\_NAME](#)

A [D3D\\_NAME](#)-typed value that identifies a predefined string that determines the functionality of certain pipeline stages.

## ComponentType

Type: [D3D\\_REGISTER\\_COMPONENT\\_TYPE](#)

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

Type: [BYTE](#)

Mask which indicates which components of a register are used.

## ReadWriteMask

Type: [BYTE](#)

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

Type: [UINT](#)

Indicates which stream the geometry shader is using for the signature parameter.

## MinPrecision

Type: [D3D\\_MIN\\_PRECISION](#)

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 [ID3D11ShaderReflection::GetInputParameterDesc](#).

## Requirements

Header	d3d11shader.h
--------	---------------

## See also

[Shader Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D11\_TRACE\_REGISTER structure (d3d11shadertracing.h)

Article 02/22/2024

Describes a trace register.

## Syntax

C++

```
typedef struct D3D11_TRACE_REGISTER {
    D3D11_TRACE_REGISTER_TYPE RegType;
    union {
        UINT16 Index1D;
        UINT16 Index2D[2];
    };
    UINT8             OperandIndex;
    UINT8             Flags;
} D3D11_TRACE_REGISTER;
```

## Members

RegType

A [D3D11\\_TRACE\\_REGISTER\\_TYPE](#)-typed value that identifies the type of register that the shader-trace object uses.

Index1D

An index for one-dimensional arrays. This index is used by the following register types:

- vertex shader or pixel shader input: v[Index1D]
- temp: r[Index1D]
- output: o[Index1D]
- immediate constant buffer: icb[Index1D]
- sampler s[Index1D]
- resource r[Index1D]
- input patch constant register: vpc[Index1D]
- unordered access view: u[Index1D]
- thread group shared memory: g[Index1D]

## Index2D[2]

An array of indexes for two-dimensional arrays. These indexes are used by the following register types:

- GS input: v[Index2D[0]][Index2D[1]]
- indexable temp: x[Index2D[0]][Index2D[1]]
- constant buffer: cb#[#]
- input control point register: vcp[Index2D[0]][Index2D[1]]
- output control point register: voctp[Index2D[0]][Index2D[1]]

## OperandIndex

The index of the operand, which starts from 0.

## Flags

A combination of the following flags that are combined by using a bitwise **OR** operation. The resulting value specifies more about the trace register.

 Expand table

Flag	Description
D3D11_TRACE_REGISTER_FLAGS_RELATIVE_INDEXING (0x1)	Access to the register is part of the relative indexing of a register.

## Remarks

The following register types do not require an index:

- input PrimitiveID
- output oDepth
- immediate32
- NULL register
- output control point ID (this is actually an input; it defines the output that the thread controls)
- input fork instance ID
- input join instance ID
- input domain point register
- cycle counter

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

# Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_TRACE\_STATS structure (d3d11shadertracing.h)

Article09/01/2022

Specifies statistics about a trace.

## Syntax

C++

```
typedef struct D3D11_TRACE_STATS {
    D3D11_SHADER_TRACE_DESC           TraceDesc;
    UINT8                             NumInvocationsInStamp;
    UINT8                             TargetStampIndex;
    UINT                             NumTraceSteps;
    D3D11_TRACE_COMPONENT_MASK        InputMask[32];
    D3D11_TRACE_COMPONENT_MASK        OutputMask[32];
    UINT16                            NumTemps;
    UINT16                            MaxIndexableTempIndex;
    UINT16                            IndexableTempSize[4096];
    UINT16                            ImmediateConstantBufferSize;
    UINT                             PixelPosition[4][2];
    UINT64                            PixelCoverageMask[4];
    UINT64                            PixelDiscardedMask[4];
    UINT64                            PixelCoverageMaskAfterShader[4];
    UINT64                            PixelCoverageMaskAfterA2CSampleMask[4];
    UINT64                            PixelCoverageMaskAfterA2CSampleMaskDepth[4];
    UINT64                            PixelCoverageMaskAfterA2CSampleMaskDepthStencil[4];
    BOOL                             PSOutputsDepth;
    BOOL                             PSOutputsMask;
    D3D11_TRACE_GS_INPUT_PRIMITIVE   GSInputPrimitive;
    BOOL                             GSInputsPrimitiveID;
    D3D11_TRACE_COMPONENT_MASK        HSOutputPatchConstantMask[32];
    D3D11_TRACE_COMPONENT_MASK        DSInputPatchConstantMask[32];
} D3D11_TRACE_STATS;
```

## Members

TraceDesc

A [D3D11\\_SHADER\\_TRACE\\_DESC](#) structure that describes the shader trace object for which this structure specifies statistics.

#### `NumInvocationsInStamp`

The number of calls in the stamp for the trace. This value is always 1 for vertex shaders, hull shaders, domain shaders, geometry shaders, and compute shaders. This value is 4 for pixel shaders.

#### `TargetStampIndex`

The index of the target stamp. This value is always 0 for vertex shaders, hull shaders, domain shaders, geometry shaders, and compute shaders. However, for pixel shaders this value indicates which of the four pixels in the stamp is the target for the trace. You can examine the traces for other pixels in the stamp to determine how derivative calculations occurred. You can make this determination by correlating the registers across traces.

#### `NumTraceSteps`

The total number of steps for the trace. This number is the same for all stamp calls.

#### `InputMask[32]`

The component trace mask for each input v# register. For information about `D3D11_TRACE_COMPONENT_MASK`, see [D3D11\\_TRACE\\_VALUE](#).

For vertex shaders, geometry shaders, pixel shaders, hull shaders, and domain shaders, the valid range is [0..31]. For compute shaders, this member is not applicable. Also, inputs for geometry shaders are 2D-indexed. For example, consider `v[vertex][attribute]`. In this example, the range of `[attribute]` is [0..31]. The `[vertex]` axis is the same size for all inputs, which are determined by the `GSIInputPrimitive` member.

Similarly, inputs for hull shader and domain shader are 2D-indexed. For example, consider `v[vertex][attribute]`. In this example, the range of `[attribute]` is [0..15]. The `[vertex]` axis is the same size for all inputs.

#### `OutputMask[32]`

The component trace mask for each output o# register. For information about `D3D11_TRACE_COMPONENT_MASK`, see [D3D11\\_TRACE\\_VALUE](#).

For vertex shaders and geometry shaders, the valid range is [0..31]. For pixel shaders, the valid range is [0..7]. For compute shaders, this member is not applicable. For output control points for hull shaders, the registers are 2D-indexed. For example, consider `ocp[vertex][attribute]`. In this example, the range of `[attribute]` is [0..31]. The `[vertex]` axis is the same size for all inputs.

`NumTemps`

The number of temps, that is, 4x32 bit r# registers that are declared.

`MaxIndexableTempIndex`

The maximum index #+1 of all indexable temps `x#[ ]` that are declared. If they are declared sparsely (for example, `x3[12]` and `x200[30]` only), this value is 201 (200+1).

`IndexableTempSize[4096]`

The number of temps for each indexable temp `x#[numTemps]`. You can only have temps up to the value in the `MaxIndexableTempIndex` member.

`ImmediateConstantBufferSize`

The number of 4x32 bit values (if any) that are in the immediate constant buffer.

`PixelPosition[4]`

`PixelCoverageMask[4]`

**Note** This member is for pixel shaders only, [stampIndex].

A mask that indicates which MSAA samples are covered for each stamp. This coverage occurs before alpha-to-coverage, depth, and stencil operations are performed on the pixel. For non-MSAA, examine the least significant bit (LSB). This mask can be 0 for pixels that are only executed to support derivatives for neighboring pixels.

`PixelDiscardedMask[4]`

**Note** This member is for pixel shaders only, [stampIndex].

A mask that indicates discarded samples. If the pixel shader runs at pixel-frequency, "discard" turns off all the samples. If all the samples are off, the following four mask members are also 0.

`PixelCoverageMaskAfterShader[4]`

**Note** This member is for pixel shaders only, [stampIndex].

A mask that indicates the MSAA samples that are covered. For non-MSAA, examine the LSB.

`PixelCoverageMaskAfterA2CSampleMask[4]`

**Note** This member is for pixel shaders only, [stampIndex].

A mask that indicates the MSAA samples that are covered after alpha-to-coverage+sampleMask, but before depth and stencil. For non-MSAA, examine the LSB.

`PixelCoverageMaskAfterA2CSampleMaskDepth[4]`

**Note** This member is for pixel shaders only, [stampIndex].

A mask that indicates the MSAA samples that are covered after alpha-to-coverage+sampleMask+depth, but before stencil. For non-MSAA, examine the LSB.

`PixelCoverageMaskAfterA2CSampleMaskDepthStencil[4]`

**Note** This member is for pixel shaders only, [stampIndex].

A mask that indicates the MSAA samples that are covered after alpha-to-coverage+sampleMask+depth+stencil. For non-MSAA, examine the LSB.

`PSOutputsDepth`

A value that specifies whether this trace is for a pixel shader that outputs the oDepth register. TRUE indicates that the pixel shader outputs the oDepth register; otherwise, FALSE.

`PSOutputsMask`

A value that specifies whether this trace is for a pixel shader that outputs the oMask register. TRUE indicates that the pixel shader outputs the oMask register; otherwise, FALSE.

`GSIInputPrimitive`

A [D3D11\\_TRACE\\_GS\\_INPUT\\_PRIMITIVE](#)-typed value that identifies the type of geometry shader input primitive. That is, this value identifies: {point, line, triangle, line\_adj, triangle\_adj} or the number of vertices: 1, 2, 3, 4, or 6 respectively. For example, for a

line, input v[][#] is actually v[2][#]. For vertex shaders and pixel shaders, set this member to [D3D11\\_TRACE\\_GS\\_INPUT\\_PRIMITIVE\\_UNDEFINED](#).

#### `GSIinputsPrimitiveID`

A value that specifies whether this trace is for a geometry shader that inputs the PrimitiveID register. TRUE indicates that the geometry shader inputs the PrimitiveID register; otherwise, FALSE.

#### `HSOutputPatchConstantMask[32]`

**Note** This member is for hull shaders only.

The component trace mask for the hull-shader output. For information about D3D11\_TRACE\_COMPONENT\_MASK, see [D3D11\\_TRACE\\_VALUE](#).

The [D3D11\\_TRACE\\_INPUT\\_PRIMITIVE\\_ID\\_REGISTER](#) value is available through a call to the [ID3D11ShaderTrace::GetInitialRegisterContents](#) method.

#### `DSInputPatchConstantMask[32]`

**Note** This member is for domain shaders only.

The component trace mask for the domain-shader input. For information about D3D11\_TRACE\_COMPONENT\_MASK, see [D3D11\\_TRACE\\_VALUE](#).

The following values are available through a call to the [ID3D11ShaderTrace::GetInitialRegisterContents](#) method:

- [D3D11\\_TRACE\\_INPUT\\_PRIMITIVE\\_ID\\_REGISTER](#)
- [D3D11\\_TRACE\\_INPUT\\_DOMAIN\\_POINT\\_REGISTER](#)

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[ID3D11ShaderTrace::GetTraceStats](#)

[Shader Structures](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# D3D11\_TRACE\_STEP structure (d3d11shadertracing.h)

Article 06/24/2021

Describes a trace step, which is an instruction.

## Syntax

C++

```
typedef struct D3D11_TRACE_STEP {
    UINT                         ID;
    BOOL                        InstructionActive;
    UINT8                       NumRegistersWritten;
    UINT8                       NumRegistersRead;
    D3D11_TRACE_MISC_OPERATIONS_MASK MiscOperations;
    UINT                        OpcodeType;
    UINT64                      CurrentGlobalCycle;
} D3D11_TRACE_STEP;
```

## Members

### ID

A number that identifies the instruction, as an offset into the executable instructions that are present in the shader.

HLSL debugging information uses the same convention. Therefore, HLSL instructions are matched to a set of IDs. You can then map an ID to a disassembled string that can be displayed to the user.

### InstructionActive

A value that specifies whether the instruction is active. This value is TRUE if something happened; therefore, you should parse other data in this structure. Otherwise, nothing happened; for example, if an instruction is disabled due to flow control even though other pixels in the stamp execute it.

### NumRegistersWritten

The number of registers for the instruction that are written to. The range of registers is [0...NumRegistersWritten-1]. You can pass a register number to the *writtenRegisterIndex*

parameter of [ID3D11ShaderTrace::GetWrittenRegister](#) to retrieve individual write-register information.

#### NumRegistersRead

The number of registers for the instruction that are read from. The range of registers is [0...NumRegistersRead-1]. You can pass a register number to the *readRegisterIndex* parameter of [ID3D11ShaderTrace::GetReadRegister](#) to retrieve individual read-register information.

#### MiscOperations

A combination of the following values that are combined by using a bitwise **OR** operation. The resulting value specifies the mask for the trace miscellaneous operations. These flags indicate the possible effect of a shader operation when it does not write any output registers. For example, the "add r0, r1 ,r2" operation writes to the r0 register; therefore, you can look at the trace-written register's information to determine what the operation changed. However, some shader instructions do not write any registers, but still effect those registers.

Flag	Description
D3D11_TRACE_MISC_GS_EMIT (0x1)	The operation was a geometry shader data emit.
D3D11_TRACE_MISC_GS_CUT (0x2)	The operation was a geometry shader strip cut.
D3D11_TRACE_MISC_PS_DISCARD (0x4)	The operation was a pixel shader discard, which rejects the pixel.
D3D11_TRACE_MISC_GS_EMIT_STREAM (0x8)	Same as D3D11_TRACE_MISC_GS_EMIT, except in <a href="#">shader model 5</a> where you can specify a particular stream to emit to.
D3D11_TRACE_MISC_GS_CUT_STREAM (0x10)	Same as D3D11_TRACE_MISC_GS_CUT, except in <a href="#">shader model 5</a> where you can specify a particular stream to strip cut.
D3D11_TRACE_MISC_HALT (0x20)	The operation was a shader halt instruction, which stops shader execution. The HLSL <a href="#">abort</a> intrinsic function causes a halt.
D3D11_TRACE_MISC_MESSAGE (0x40)	The operation was a shader message output, which can be logged to the information queue. The HLSL <a href="#">printf</a> and <a href="#">errorf</a> intrinsic functions cause messages.

If the **NumRegistersWritten** member is 0, examine this member although this member still might be empty (0).

#### OpcodeType

A number that specifies the type of instruction (for example, [add](#), [mul](#), and so on). You can ignore this member if you do not know the number for the instruction type. This member offers a minor convenience at the cost of bloating the trace slightly. You can use the **ID** member and map back to the original shader code to retrieve the full information about the instruction.

#### CurrentGlobalCycle

The global cycle count for this step. You can use this member to correlate parallel thread execution via multiple simultaneous traces, for example, for the compute shader.

**Note** Multiple threads at the same point in execution might log the same **CurrentGlobalCycle**.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3D11\_TRACE\_VALUE structure (d3d11shadertracing.h)

Article 02/22/2024

Describes a trace value.

## Syntax

C++

```
typedef struct D3D11_TRACE_VALUE {
    UINT Bits[4];
    D3D11_TRACE_COMPONENT_MASK ValidMask;
} D3D11_TRACE_VALUE;
```

## Members

Bits[4]

An array of bits that make up the trace value. The [0] element is X.

**Note** This member can hold **float**, **UINT**, or **INT** data. The elements are specified as **UINT** rather than using a union to minimize the risk of x86 SNaN->QNaN quashing during float assignment. If the bits are displayed, they can be interpreted as **float** at the last moment.

ValidMask

A combination of the following component values that are combined by using a bitwise OR operation. The resulting value specifies the component trace mask.

[+] Expand table

Flag	Description
D3D11_TRACE_COMPONENT_X (0x1)	The x component of the trace mask.
D3D11_TRACE_COMPONENT_Y (0x2)	The y component of the trace mask.

D3D11_TRACE_COMPONENT_Z (0x4)	The depth z component of the trace mask.
D3D11_TRACE_COMPONENT_W (0x8)	The depth w component of the trace mask.

Ignore unmasked values, particularly if deltas are accumulated.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_VERTEX\_SHADER\_TRACE\_DESC structure (d3d11shadertracing.h)

Article02/22/2024

Describes an instance of a vertex shader to trace.

## Syntax

C++

```
typedef struct D3D11_VERTEX_SHADER_TRACE_DESC {
    UINT64 Invocation;
} D3D11_VERTEX_SHADER_TRACE_DESC;
```

## Members

### Invocation

The invocation number of the instance of the vertex shader.

## Remarks

This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Structures](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# Shader Enumerations (Direct3D 11 Graphics)

Article • 12/10/2020

Enumerations are used to specify information about shaders.

## In this section

Topic	Description
<a href="#">D3D11_CBUFFER_TYPE</a>	Indicates a constant buffer's type.
<a href="#">D3D_PARAMETER_FLAGS</a>	Indicates semantic flags for function parameters.
<a href="#">D3D11_RESOURCE_RETURN_TYPE</a>	Indicates return value type.
<a href="#">D3D11_SHADER_TYPE</a>	Identifies a shader type for tracing.
<a href="#">D3D11_SHADER_VERSION_TYPE</a>	Indicates shader type.
<a href="#">D3D11_TESSELLATOR_DOMAIN</a>	Domain options for tessellator data.
<a href="#">D3D11_TESSELLATOR_PARTITIONING</a>	Partitioning options.
<a href="#">D3D11_TESSELLATOR_OUTPUT_PRIMITIVE</a>	Output primitive types.
<a href="#">D3D11_TRACE_GS_INPUT_PRIMITIVE</a>	Identifies the type of geometry shader input primitive.
<a href="#">D3D11_TRACE_REGISTER_TYPE</a>	Identifies a type of trace register.

## Related topics

[Shader Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D11\_CBUFFER\_TYPE enumeration

Article 12/01/2017

Indicates a constant buffer's type.

## Syntax

```
c++  
  
typedef enum _D3D11_CBUFFER_TYPE {  
    D3D11_CT_CBUFFER,  
    D3D11_CT_TBUFFER,  
    D3D11_CT_INTERFACE_POINTERS,  
    D3D11_CT_RESOURCE_BIND_INFO  
} D3D11_CBUFFER_TYPE, *LPD3D11_CBUFFER_TYPE;
```

## Constants

- **D3D11\_CT\_CBUFFER**  
A buffer containing scalar constants.
- **D3D11\_CT\_TBUFFER**  
A buffer containing texture data.
- **D3D11\_CT\_INTERFACE\_POINTERS**  
A buffer containing interface pointers.
- **D3D11\_CT\_RESOURCE\_BIND\_INFO**  
A buffer containing binding information.

## Remarks

The D3D11\_CBUFFER\_TYPE enumeration is type defined in the D3D11Shader.h header file as a [D3D\\_CBUFFER\\_TYPE](#) enumeration, which is fully defined in the D3DCCommon.h header file.

```
typedef D3D_CBUFFER_TYPE D3D11_CBUFFER_TYPE;
```

## Requirements

[ ] [Expand table](#)

Header

D3d11shader.h

## See also

[Shader Enumerations ↗](#)

# D3D\_PARAMETER\_FLAGS enumeration (d3dcommon.h)

Article 02/22/2024

Indicates semantic flags for function parameters.

## Syntax

C++

```
typedef enum _D3D_PARAMETER_FLAGS {
    D3D_PF_NONE = 0,
    D3D_PF_IN = 0x1,
    D3D_PF_OUT = 0x2,
    D3D_PF_FORCE_DWORD = 0x7fffffff
} D3D_PARAMETER_FLAGS;
```

## Constants

[ ] [Expand table](#)

<code>D3D_PF_NONE</code>	Value: <i>0</i> The parameter has no semantic flags.
<code>D3D_PF_IN</code>	Value: <i>0x1</i> Indicates an input parameter.
<code>D3D_PF_OUT</code>	Value: <i>0x2</i> Indicates an output parameter.
<code>D3D_PF_FORCE_DWORD</code>	Value: <i>0x7fffffff</i> Forces this enumeration to compile to 32 bits in size. Without this value, some compilers would allow this enumeration to compile to a size other than 32 bits. This value is not used.

## Requirements

Expand table

Requirement	Value
Header	d3dcommon.h (include D3D11Shader.h)

## See also

[D3D11\\_PARAMETER\\_DESC](#)

[Shader Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_RESOURCE\_RETURN\_TYPE enumeration (d3dcommon.h)

Article01/31/2022

Indicates return value type.

## ⓘ Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_RESOURCE_RETURN_TYPE {
    D3D_RETURN_TYPE_UNORM = 1,
    D3D_RETURN_TYPE_SNORM = 2,
    D3D_RETURN_TYPE_SINT = 3,
    D3D_RETURN_TYPE_UINT = 4,
    D3D_RETURN_TYPE_FLOAT = 5,
    D3D_RETURN_TYPE_MIXED = 6,
    D3D_RETURN_TYPE_DOUBLE = 7,
    D3D_RETURN_TYPE_CONTINUED = 8,
    D3D10_RETURN_TYPE_UNORM,
    D3D10_RETURN_TYPE_SNORM,
    D3D10_RETURN_TYPE_SINT,
    D3D10_RETURN_TYPE_UINT,
    D3D10_RETURN_TYPE_FLOAT,
    D3D10_RETURN_TYPE_MIXED,
    D3D11_RETURN_TYPE_UNORM,
    D3D11_RETURN_TYPE_SNORM,
    D3D11_RETURN_TYPE_SINT,
    D3D11_RETURN_TYPE_UINT,
    D3D11_RETURN_TYPE_FLOAT,
    D3D11_RETURN_TYPE_MIXED,
    D3D11_RETURN_TYPE_DOUBLE,
    D3D11_RETURN_TYPE_CONTINUED
} ;
```

## Constants

[Expand table](#)

D3D\_RETURN\_TYPE\_UNORM

Value: 1

D3D\_RETURN\_TYPE\_SNORM

Value: 2

D3D\_RETURN\_TYPE\_SINT

Value: 3

D3D\_RETURN\_TYPE\_UINT

Value: 4

D3D\_RETURN\_TYPE\_FLOAT

Value: 5

D3D\_RETURN\_TYPE\_MIXED

Value: 6

D3D\_RETURN\_TYPE\_DOUBLE

Value: 7

D3D10\_RETURN\_TYPE\_CONTINUED

Value: 8

D3D10\_RETURN\_TYPE\_UNORM

D3D10\_RETURN\_TYPE\_SNORM

D3D10\_RETURN\_TYPE\_SINT

D3D10\_RETURN\_TYPE\_UINT

D3D10\_RETURN\_TYPE\_FLOAT

D3D10\_RETURN\_TYPE\_MIXED

D3D11\_RETURN\_TYPE\_UNORM

Return type is UNORM.

D3D11\_RETURN\_TYPE\_SNORM

Return type is SNORM.

D3D11\_RETURN\_TYPE\_SINT

Return type is SINT.

D3D11\_RETURN\_TYPE\_UINT

Return type is UINT.

**D3D11\_RETURN\_TYPE\_FLOAT**

Return type is FLOAT.

**D3D11\_RETURN\_TYPE\_MIXED**

Return type is unknown.

**D3D11\_RETURN\_TYPE\_DOUBLE**

Return type is DOUBLE.

**D3D11\_RETURN\_TYPE\_CONTINUED**

Return type is a multiple-dword type, such as a double or uint64, and the component is continued from the previous component that was declared. The first component represents the lower bits.

## Remarks

The `D3D11_RESOURCE_RETURN_TYPE` enumeration is type defined in the `D3D11shader.h` header file as a `D3D_RESOURCE_RETURN_TYPE` enumeration, which is fully defined in the `D3DCCommon.h` header file.

```
typedef D3D_RESOURCE_RETURN_TYPE D3D11_RESOURCE_RETURN_TYPE;
```

## Requirements

 Expand table

Requirement	Value
Header	<code>d3dcommon.h</code>

## See also

[Shader Enumerations](#)

# D3D11\_SHADER\_TYPE enumeration (d3d11shadertracing.h)

Article 01/31/2022

Identifies a shader type for tracing.

## Syntax

C++

```
typedef enum D3D11_SHADER_TYPE {
    D3D11_VERTEX_SHADER = 1,
    D3D11_HULL_SHADER = 2,
    D3D11_DOMAIN_SHADER = 3,
    D3D11_GEOMETRY_SHADER = 4,
    D3D11_PIXEL_SHADER = 5,
    D3D11_COMPUTE_SHADER = 6
};
```

## Constants

`D3D11_VERTEX_SHADER`

Value: 1

Identifies a vertex shader.

`D3D11_HULL_SHADER`

Value: 2

Identifies a hull shader.

`D3D11_DOMAIN_SHADER`

Value: 3

Identifies a domain shader.

`D3D11_GEOMETRY_SHADER`

Value: 4

Identifies a geometry shader.

`D3D11_PIXEL_SHADER`

Value: 5

Identifies a pixel shader.

D3D11\_COMPUTE\_SHADER

Value: 6

Identifies a compute shader.

## Remarks

D3D11\_SHADER\_TYPE identifies the type of shader in a [D3D11\\_SHADER\\_TRACE\\_DESC](#) structure.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D11\_SHADER\_VERSION\_TYPE enumeration (d3d11shader.h)

Article 02/22/2024

Indicates shader type.

## Syntax

C++

```
typedef enum D3D11_SHADER_VERSION_TYPE {
    D3D11_SHVER_PIXEL_SHADER = 0,
    D3D11_SHVER_VERTEX_SHADER = 1,
    D3D11_SHVER_GEOMETRY_SHADER = 2,
    D3D11_SHVER_HULL_SHADER = 3,
    D3D11_SHVER_DOMAIN_SHADER = 4,
    D3D11_SHVER_COMPUTE_SHADER = 5,
    D3D11_SHVER_RESERVED0 = 0xFFFF0
};
```

## Constants

[ ] [Expand table](#)

	<b>D3D11_SHVER_PIXEL_SHADER</b>
Value:	0
	Pixel shader.
	<b>D3D11_SHVER_VERTEX_SHADER</b>
Value:	1
	Vertex shader.
	<b>D3D11_SHVER_GEOMETRY_SHADER</b>
Value:	2
	Geometry shader.
	<b>D3D11_SHVER_HULL_SHADER</b>
Value:	3
	Hull shader.
	<b>D3D11_SHVER_DOMAIN_SHADER</b>
Value:	4

Domain shader.

D3D11\_SHVER\_COMPUTE\_SHADER

Value: 5

Compute shader.

D3D11\_SHVER\_RESERVED0

Value: 0xFFFF0

Indicates the end of the enumeration constants.

  Expand table

Requirement	Value
Header	d3d11shader.h

## See also

[Shader Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_TESSELLATOR\_DOMAIN enumeration (d3dcommon.h)

Article 02/22/2024

Domain options for tessellator data.

## Syntax

C++

```
typedef enum D3D_TESSELLATOR_DOMAIN {
    D3D_TESSELLATOR_DOMAIN_UNDEFINED = 0,
    D3D_TESSELLATOR_DOMAIN_ISOLINE = 1,
    D3D_TESSELLATOR_DOMAIN_TRI = 2,
    D3D_TESSELLATOR_DOMAIN_QUAD = 3,
    D3D11_TESSELLATOR_DOMAIN_UNDEFINED,
    D3D11_TESSELLATOR_DOMAIN_ISOLINE,
    D3D11_TESSELLATOR_DOMAIN_TRI,
    D3D11_TESSELLATOR_DOMAIN_QUAD
} ;
```

## Constants

[] Expand table

D3D_TESSELLATOR_DOMAIN_UNDEFINED
Value: 0
D3D_TESSELLATOR_DOMAIN_ISOLINE
Value: 1
D3D_TESSELLATOR_DOMAIN_TRI
Value: 2
D3D_TESSELLATOR_DOMAIN_QUAD
Value: 3
D3D11_TESSELLATOR_DOMAIN_UNDEFINED
The data type is undefined.
D3D11_TESSELLATOR_DOMAIN_ISOLINE
IsoLine data.

D3D11\_TESSELLATOR\_DOMAIN\_TRI

Triangle data.

D3D11\_TESSELLATOR\_DOMAIN\_QUAD

Quad data.

## Remarks

The data domain defines the type of data. This enumeration is used by [D3D11\\_SHADER\\_DESC](#).

The **D3D11\_TESSELLATOR\_DOMAIN** enumeration is type defined in the D3D11Shader.h header file as a [D3D\\_TESSELLATOR\\_DOMAIN](#) enumeration, which is fully defined in the D3DCCommon.h header file.

```
typedef D3D_TESSELLATOR_DOMAIN D3D11_TESSELLATOR_DOMAIN;
```

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Shader Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_TESSELLATOR\_PARTITIONING enumeration (d3dcommon.h)

Article 02/22/2024

Partitioning options.

## Syntax

C++

```
typedef enum D3D_TESSELLATOR_PARTITIONING {
    D3D_TESSELLATOR_PARTITIONING_UNDEFINED = 0,
    D3D_TESSELLATOR_PARTITIONING_INTEGER = 1,
    D3D_TESSELLATOR_PARTITIONING_POW2 = 2,
    D3D_TESSELLATOR_PARTITIONING_FRACTIONAL_ODD = 3,
    D3D_TESSELLATOR_PARTITIONING_FRACTIONAL_EVEN = 4,
    D3D11_TESSELLATOR_PARTITIONING_UNDEFINED,
    D3D11_TESSELLATOR_PARTITIONING_INTEGER,
    D3D11_TESSELLATOR_PARTITIONING_POW2,
    D3D11_TESSELLATOR_PARTITIONING_FRACTIONAL_ODD,
    D3D11_TESSELLATOR_PARTITIONING_FRACTIONAL_EVEN
} ;
```

## Constants

[ ] Expand table

<code>D3D_TESSELLATOR_PARTITIONING_UNDEFINED</code>
Value: 0
<code>D3D_TESSELLATOR_PARTITIONING_INTEGER</code>
Value: 1
<code>D3D_TESSELLATOR_PARTITIONING_POW2</code>
Value: 2
<code>D3D_TESSELLATOR_PARTITIONING_FRACTIONAL_ODD</code>
Value: 3
<code>D3D_TESSELLATOR_PARTITIONING_FRACTIONAL_EVEN</code>
Value: 4

#### D3D11\_TESSELLATOR\_PARTITIONING\_UNDEFINED

The partitioning type is undefined.

#### D3D11\_TESSELLATOR\_PARTITIONING\_INTEGER

Partition with integers only.

#### D3D11\_TESSELLATOR\_PARTITIONING\_POW2

Partition with a power-of-two number only.

#### D3D11\_TESSELLATOR\_PARTITIONING\_FRACTIONAL\_ODD

Partition with an odd, fractional number.

#### D3D11\_TESSELLATOR\_PARTITIONING\_FRACTIONAL\_EVEN

Partition with an even, fractional number.

## Remarks

During tessellation, the partition option helps to determine how the algorithm chooses the next partition value; this enumeration is used by [D3D11\\_SHADER\\_DESC](#).

The **D3D11\_TESSELLATOR\_PARTITIONING** enumeration is type defined in the D3D11Shader.h header file as a [D3D\\_TESSELLATOR\\_PARTITIONING](#) enumeration, which is fully defined in the D3DCCommon.h header file.

```
typedef D3D_TESSELLATOR_PARTITIONING D3D11_TESSELLATOR_PARTITIONING;
```

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Shader Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_TESSELLATOR\_OUTPUT\_PRIMITIVE enumeration (d3dcommon.h)

Article 02/22/2024

Output primitive types.

## Syntax

C++

```
typedef enum D3D_TESSELLATOR_OUTPUT_PRIMITIVE {
    D3D_TESSELLATOR_OUTPUT_UNDEFINED = 0,
    D3D_TESSELLATOR_OUTPUT_POINT = 1,
    D3D_TESSELLATOR_OUTPUT_LINE = 2,
    D3D_TESSELLATOR_OUTPUT_TRIANGLE_CW = 3,
    D3D_TESSELLATOR_OUTPUT_TRIANGLE_CCW = 4,
    D3D11_TESSELLATOR_OUTPUT_UNDEFINED,
    D3D11_TESSELLATOR_OUTPUT_POINT,
    D3D11_TESSELLATOR_OUTPUT_LINE,
    D3D11_TESSELLATOR_OUTPUT_TRIANGLE_CW,
    D3D11_TESSELLATOR_OUTPUT_TRIANGLE_CCW
} ;
```

## Constants

[+] Expand table

<code>D3D_TESSELLATOR_OUTPUT_UNDEFINED</code>
Value: 0
<code>D3D_TESSELLATOR_OUTPUT_POINT</code>
Value: 1
<code>D3D_TESSELLATOR_OUTPUT_LINE</code>
Value: 2
<code>D3D_TESSELLATOR_OUTPUT_TRIANGLE_CW</code>
Value: 3
<code>D3D_TESSELLATOR_OUTPUT_TRIANGLE_CCW</code>
Value: 4

#### D3D11\_TESSELLATOR\_OUTPUT\_UNDEFINED

The output primitive type is undefined.

#### D3D11\_TESSELLATOR\_OUTPUT\_POINT

The output primitive type is a point.

#### D3D11\_TESSELLATOR\_OUTPUT\_LINE

The output primitive type is a line.

#### D3D11\_TESSELLATOR\_OUTPUT\_TRIANGLE\_CW

The output primitive type is a clockwise triangle.

#### D3D11\_TESSELLATOR\_OUTPUT\_TRIANGLE\_CCW

The output primitive type is a counter clockwise triangle.

## Remarks

The output primitive type determines how the tessellator output data is organized; this enumeration is used by [D3D11\\_SHADER\\_DESC](#).

The **D3D11\_TESSELLATOR\_OUTPUT\_PRIMITIVE** enumeration is type defined in the D3D11Shader.h header file as a [D3D\\_TESSELLATOR\\_OUTPUT\\_PRIMITIVE](#) enumeration, which is fully defined in the D3DCCommon.h header file.

```
typedef D3D_TESSELLATOR_OUTPUT_PRIMITIVE D3D11_TESSELLATOR_OUTPUT_PRIMITIVE;
```

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Shader Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_TRACE\_GS\_INPUT\_PRIMITIVE enumeration (d3d11shadertracing.h)

Article 02/22/2024

Identifies the type of geometry shader input primitive.

## Syntax

C++

```
typedef enum D3D11_TRACE_GS_INPUT_PRIMITIVE {
    D3D11_TRACE_GS_INPUT_PRIMITIVE_UNDEFINED = 0,
    D3D11_TRACE_GS_INPUT_PRIMITIVE_POINT = 1,
    D3D11_TRACE_GS_INPUT_PRIMITIVE_LINE = 2,
    D3D11_TRACE_GS_INPUT_PRIMITIVE_TRIANGLE = 3,
    D3D11_TRACE_GS_INPUT_PRIMITIVE_LINE_ADJ = 6,
    D3D11_TRACE_GS_INPUT_PRIMITIVE_TRIANGLE_ADJ = 7
};
```

## Constants

[+] Expand table

D3D11_TRACE_GS_INPUT_PRIMITIVE_UNDEFINED
--

Value: 0

Identifies the geometry shader input primitive as undefined.

D3D11_TRACE_GS_INPUT_PRIMITIVE_POINT
--------------------------------------

Value: 1

Identifies the geometry shader input primitive as a point.

D3D11_TRACE_GS_INPUT_PRIMITIVE_LINE
-------------------------------------

Value: 2

Identifies the geometry shader input primitive as a line.

D3D11_TRACE_GS_INPUT_PRIMITIVE_TRIANGLE
---

Value: 3

Identifies the geometry shader input primitive as a triangle.

D3D11_TRACE_GS_INPUT_PRIMITIVE_LINE_ADJ
---

Value: 6

Identifies the geometry shader input primitive as an adjacent line.

D3D11\_TRACE\_GS\_INPUT\_PRIMITIVE\_TRIANGLE\_ADJ

Value: 7

Identifies the geometry shader input primitive as an adjacent triangle.

## Remarks

D3D11\_TRACE\_GS\_INPUT\_PRIMITIVE identifies the type of geometry shader input primitive in a [D3D11\\_TRACE\\_STATS](#) structure.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	d3d11shadertracing.h

## See also

[Shader Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D11\_TRACE\_REGISTER\_TYPE enumeration (d3d11shadertracing.h)

Article01/31/2022

Identifies a type of trace register.

## Syntax

C++

```
typedef enum D3D11_TRACE_REGISTER_TYPE {
    D3D11_TRACE_OUTPUT_NULL_REGISTER = 0,
    D3D11_TRACE_INPUT_REGISTER,
    D3D11_TRACE_INPUT_PRIMITIVE_ID_REGISTER,
    D3D11_TRACE_IMMEDIATE_CONSTANT_BUFFER,
    D3D11_TRACE_TEMP_REGISTER,
    D3D11_TRACE_INDEXABLE_TEMP_REGISTER,
    D3D11_TRACE_OUTPUT_REGISTER,
    D3D11_TRACE_OUTPUT_DEPTH_REGISTER,
    D3D11_TRACE_CONSTANT_BUFFER,
    D3D11_TRACE_IMMEDIATE32,
    D3D11_TRACE_SAMPLER,
    D3D11_TRACE_RESOURCE,
    D3D11_TRACE_RASTERIZER,
    D3D11_TRACE_OUTPUT_COVERAGE_MASK,
    D3D11_TRACE_STREAM,
    D3D11_TRACE_THIS_POINTER,
    D3D11_TRACE_OUTPUT_CONTROL_POINT_ID_REGISTER,
    D3D11_TRACE_INPUT_FORK_INSTANCE_ID_REGISTER,
    D3D11_TRACE_INPUT_JOIN_INSTANCE_ID_REGISTER,
    D3D11_TRACE_INPUT_CONTROL_POINT_REGISTER,
    D3D11_TRACE_OUTPUT_CONTROL_POINT_REGISTER,
    D3D11_TRACE_INPUT_PATCH_CONSTANT_REGISTER,
    D3D11_TRACE_INPUT_DOMAIN_POINT_REGISTER,
    D3D11_TRACE_UNORDERED_ACCESS_VIEW,
    D3D11_TRACE_THREAD_GROUP_SHARED_MEMORY,
    D3D11_TRACE_INPUT_THREAD_ID_REGISTER,
    D3D11_TRACE_INPUT_THREAD_GROUP_ID_REGISTER,
    D3D11_TRACE_INPUT_THREAD_ID_IN_GROUP_REGISTER,
    D3D11_TRACE_INPUT_COVERAGE_MASK_REGISTER,
    D3D11_TRACE_INPUT_THREAD_ID_IN_GROUP_FLATTENED_REGISTER,
    D3D11_TRACE_INPUT_GS_INSTANCE_ID_REGISTER,
    D3D11_TRACE_OUTPUT_DEPTH_GREATER_EQUAL_REGISTER,
    D3D11_TRACE_OUTPUT_DEPTH_LESS_EQUAL_REGISTER,
    D3D11_TRACE_IMMEDIATE64,
    D3D11_TRACE_INPUT_CYCLE_COUNTER_REGISTER,
    D3D11_TRACE_INTERFACE_POINTER
} ;
```

# Constants

D3D11_TRACE_OUTPUT_NULL_REGISTER
Value: 0
Output <b>NULL</b> register.
D3D11_TRACE_INPUT_REGISTER
Input register.
D3D11_TRACE_INPUT_PRIMITIVE_ID_REGISTER
Input primitive ID register.
D3D11_TRACE_IMMEDIATE_CONSTANT_BUFFER
Immediate constant buffer.
D3D11_TRACE_TEMP_REGISTER
Temporary register.
D3D11_TRACE_INDEXABLE_TEMP_REGISTER
Temporary register that can be indexed.
D3D11_TRACE_OUTPUT_REGISTER
Output register.
D3D11_TRACE_OUTPUT_DEPTH_REGISTER
Output oDepth register.
D3D11_TRACE_CONSTANT_BUFFER
Constant buffer.
D3D11_TRACE_IMMEDIATE32
Immediate32 register.
D3D11_TRACE_SAMPLER
Sampler.
D3D11_TRACE_RESOURCE
Resource.
D3D11_TRACE_RASTERIZER
Rasterizer.
D3D11_TRACE_OUTPUT_COVERAGE_MASK
Output coverage mask.
D3D11_TRACE_STREAM
Stream.

`D3D11_TRACE_THIS_POINTER`

This pointer.

`D3D11_TRACE_OUTPUT_CONTROL_POINT_ID_REGISTER`

Output control point ID register (this is actually an input; it defines the output that the thread controls).

`D3D11_TRACE_INPUT_FORK_INSTANCE_ID_REGISTER`

Input fork instance ID register.

`D3D11_TRACE_INPUT_JOIN_INSTANCE_ID_REGISTER`

Input join instance ID register.

`D3D11_TRACE_INPUT_CONTROL_POINT_REGISTER`

Input control point register.

`D3D11_TRACE_OUTPUT_CONTROL_POINT_REGISTER`

Output control point register.

`D3D11_TRACE_INPUT_PATCH_CONSTANT_REGISTER`

Input patch constant register.

`D3D11_TRACE_INPUT_DOMAIN_POINT_REGISTER`

Input domain point register.

`D3D11_TRACE_UNORDERED_ACCESS_VIEW`

Unordered-access view.

`D3D11_TRACE_THREAD_GROUP_SHARED_MEMORY`

Thread group shared memory.

`D3D11_TRACE_INPUT_THREAD_ID_REGISTER`

Input thread ID register.

`D3D11_TRACE_INPUT_THREAD_GROUP_ID_REGISTER`

Thread group ID register.

`D3D11_TRACE_INPUT_THREAD_ID_IN_GROUP_REGISTER`

Input thread ID in-group register.

`D3D11_TRACE_INPUT_COVERAGE_MASK_REGISTER`

Input coverage mask register.

`D3D11_TRACE_INPUT_THREAD_ID_IN_GROUP_FLATTENED_REGISTER`

Input thread ID in-group flattened register.

`D3D11_TRACE_INPUT_GS_INSTANCE_ID_REGISTER`

Input geometry shader (GS) instance ID register.

**D3D11\_TRACE\_OUTPUT\_DEPTH\_GREATER\_EQUAL\_REGISTER**

Output oDepth greater than or equal register.

**D3D11\_TRACE\_OUTPUT\_DEPTH\_LESS\_EQUAL\_REGISTER**

Output oDepth less than or equal register.

**D3D11\_TRACE\_IMMEDIATE64**

Immediate64 register.

**D3D11\_TRACE\_INPUT\_CYCLE\_COUNTER\_REGISTER**

Cycle counter register.

**D3D11\_TRACE\_INTERFACE\_POINTER**

Interface pointer.

## Remarks

**D3D11\_TRACE\_REGISTER\_TYPE** identifies the type of trace register in a [D3D11\\_TRACE\\_REGISTER](#) structure.

**Note** This API requires the Windows Software Development Kit (SDK) for Windows 8.

## Requirements

**Minimum supported client** Windows 8 [desktop apps only]

**Minimum supported server** Windows Server 2012 [desktop apps only]

**Header** d3d11shadertracing.h

## See also

[Shader Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# 10Level9 Reference

Article • 11/04/2020

This section specifies the differences between each 10Level9 feature level and the D3D\_FEATURE\_LEVEL\_11\_0 and higher feature level for the [ID3D11Device](#) and [ID3D11DeviceContext](#) methods.

## In this section

Topic	Description
<a href="#">10Level9</a> <a href="#">ID3D11Device</a> Methods	This section lists the differences between each 10Level9 feature level and the D3D_FEATURE_LEVEL_11_0 and higher feature level for the <a href="#">ID3D11Device</a> methods.
<a href="#">10Level9</a> <a href="#">ID3D11DeviceContext</a> Methods	This section lists the differences between each 10Level9 feature level and the D3D_FEATURE_LEVEL_11_0 and higher feature level for the <a href="#">ID3D11DeviceContext</a> methods.

## Related topics

[Direct3D 11 Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# 10Level9 ID3D11Device Methods

Article • 08/19/2021

This section lists the differences between each 10Level9 feature level and the D3D\_FEATURE\_LEVEL\_11\_0 and higher feature level for the [ID3D11Device](#) methods.

- [ID3D11Device::CheckCounter](#)
- [ID3D11Device::CheckFormatSupport](#)
- [ID3D11Device::CheckMultisampleQualityLevels](#)
- [ID3D11Device::CreateBlendState](#)
- [ID3D11Device::CreateBlendState1](#)
- [ID3D11Device::CreateBuffer](#)
- [ID3D11Device::CreateCounter](#)
- [ID3D11Device::CreateDepthStencilView](#)
- [ID3D11Device::CreateDomainShader](#)
- [ID3D11Device::CreateGeometryShader](#)
- [ID3D11Device::CreateGeometryShaderWithStreamOutput](#)
- [ID3D11Device::CreateHullShader](#)
- [ID3D11Device::CreateInputLayout](#)
- [ID3D11Device::CreatePixelShader](#)
- [ID3D11Device::CreatePredicate](#)
- [ID3D11Device::CreateQuery](#)
- [ID3D11Device::CreateRasterizerState](#)
- [ID3D11Device::CreateRenderTargetView](#)
- [ID3D11Device::CreateSamplerState](#)
- [ID3D11Device::CreateShaderResourceView](#)
- [ID3D11Device::CreateTexture1D](#)
- [ID3D11Device::CreateTexture2D](#)
- [ID3D11Device::CreateTexture3D](#)
- [ID3D11Device::CreateUnorderedAccessView](#)
- [ID3D11Device::CreateVertexShader](#)
- [ID3D11Device::OpenSharedResource](#)
- [Related topics](#)

## ID3D11Device::CheckCounter

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Device-dependent counters are optionally supported. Use <a href="#">ID3D11Device::CheckCounterInfo</a> to determine support. <del> \${REMOVE} \$</del>
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CheckFormatSupport

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	See format support by <a href="#">feature level</a> \${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CheckMultisampleQualityLevels

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Feature levels make no guarantees concerning MSAA support. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateBlendState

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	AlphaToCoverageEnable must be <b>FALSE</b> . The first four BlendEnables must all have the same value. D3D11_BLEND_SRC_ALPHASAT not supported. Dual-source color blend not supported (any SrcBlend or DestBlend with SRC1 in the name)
D3D_FEATURE_LEVEL_9_2	AlphaToCoverageEnable must be <b>FALSE</b> . The first four BlendEnables must all have the same value. The first four RenderTargetWriteMasks must all have the same value. D3D11_BLEND_SRC_ALPHASAT not supported. Dual-source color blend not supported (any SrcBlend or DestBlend with SRC1 in the name)
D3D_FEATURE_LEVEL_9_3	AlphaToCoverageEnable must be <b>FALSE</b> . The first four BlendEnables must all have the same value. D3D11_BLEND_SRC_ALPHASAT not supported. Dual-source color blend not supported (any SrcBlend or DestBlend with SRC1 in the name)
D3D_FEATURE_LEVEL_10_0	Adds alpha-to-coverage

## ID3D11Device::CreateBlendState1

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Unsupported

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_2	Unsupported
D3D_FEATURE_LEVEL_9_3	Unsupported
D3D_FEATURE_LEVEL_10_0	The <i>OutputMergerLogicOp</i> member has been added to <a href="#">D3D11_FEATURE_DATA_D3D11_OPTIONS</a> , to determine support for logical operations (bitwise logic operations between pixel shader output and render target contents, refer to <a href="#">D3D11_RENDER_TARGET_BLEND_DESC1</a> ).

## ID3D11Device::CreateBuffer

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Buffers can't have render target views. Buffers must have exactly one of D3D11_BIND_VERTEX_BUFFER, D3D11_BIND_INDEX_BUFFER, or D3D11_BIND_CONSTANT_BUFFER. Only allows index buffers with the DXGI_FORMAT_R16_UINT format.
D3D_FEATURE_LEVEL_9_2	Buffers can't have render target views. Buffers must have exactly one of D3D11_BIND_VERTEX_BUFFER, D3D11_BIND_INDEX_BUFFER, or D3D11_BIND_CONSTANT_BUFFER. Allows index buffers with the DXGI_FORMAT_R16_UINT and DXGI_FORMAT_R32_UINT formats like D3D_FEATURE_LEVEL_10_0 and higher. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateCounter

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateDepthStencilView

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Does not support two-sided stencil.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateDomainShader

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11Device::CreateGeometryShader

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateGeometryShaderWithStreamOutput

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateHullShader

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11Device::CreateInputLayout

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Does not support D3D11_INPUT_PER_INSTANCE_DATA
D3D_FEATURE_LEVEL_9_2	Does not support D3D11_INPUT_PER_INSTANCE_DATA
D3D_FEATURE_LEVEL_9_3	Vertex stream zero must have D3D11_INPUT_PER_VERTEX_DATA, if any streams have D3D11_INPUT_PER_VERTEX_DATA

See the format support by [feature level chart](#) for details on what formats can be used for vertex data at each feature level.

## ID3D11Device::CreatePixelShader

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Must use ps_4_0_level_9_1
D3D_FEATURE_LEVEL_9_2	Must use ps_4_0_level_9_1
D3D_FEATURE_LEVEL_9_3	Must use ps_4_0_level_9_3 or ps_4_0_level_9_1

## ID3D11Device::CreatePredicate

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level. <del>{REMOVE}</del>
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateQuery

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Event queries are supported. Timestamp queries are optional: call <a href="#">CreateQuery</a> to determine support.
D3D_FEATURE_LEVEL_9_2	Event and occlusion queries are supported. Timestamp queries are optional: call <a href="#">CreateQuery</a> to determine support.
D3D_FEATURE_LEVEL_9_3	Event and occlusion queries are supported. Timestamp queries are optional: call <a href="#">CreateQuery</a> to determine support.

## ID3D11Device::CreateRasterizerState

Feature Level	Behavior Differences

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	DepthClipEnable must be TRUE. DepthBiasClamp must be set to 0. <del>{REMOVE}</del>
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateRenderTargetView

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Can only support render target views of Texture2D objects. <del>{REMOVE}</del>
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateSamplerState

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Comparison filter is not supported. Border color must be within [0,1] Min LOD cannot be fractional Max LOD must be FLT_MAX Maximum anisotropy is 2. D3D11_TEXTURE_ADDRESS_MIRRORONCE not supported.
D3D_FEATURE_LEVEL_9_2	Comparison filter is not supported. Border color must be within [0,1] Min LOD cannot be fractional Max LOD must be FLT_MAX Maximum anisotropy is 16. <del>{REMOVE}</del>
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateShaderResourceView

Feature Level	MostDetailedMip plus MipLevels must include lowest LOD (smallest subresource)	View must include all resource array elements
D3D_FEATURE_LEVEL_9_1	Yes	yes
D3D_FEATURE_LEVEL_9_2	Yes	Yes
D3D_FEATURE_LEVEL_9_3	Yes	Yes

## ID3D11Device::CreateTexture1D

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level. <del>{REMOVE}</del>
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateTexture2D

Texture2D resources have limits on their width and height that differ across [feature levels](#). In feature levels 9\_3, the following are guaranteed minima, and individual implementations may exceed the requirements.

Feature Level	If MipCount > 1, Dimensions must be integral power of two	Minimum supported texture dimension	Cube textures dimensions must be power of two	If MISC_TEXTURECUBE is set, the ArraySize is:	If MISC_TEXTURECUBE is not set, the ArraySize is:
D3D_FEATURE_LEVEL_9_1	Yes	2048	Yes	6	1
D3D_FEATURE_LEVEL_9_2	Yes	2048	Yes	6	1
D3D_FEATURE_LEVEL_9_3	Yes	4096	Yes	6	1

In the previous table, the full name of MISC\_TEXTURECUBE is [D3D11\\_RESOURCE\\_MISC\\_TEXTURECUBE](#).

The following are true for all 9\_\* [feature levels](#):

- When using D3D11\_USAGE\_DEFAULT or D3D11\_USAGE\_IMMUTABLE, BindFlags cannot be zero.
- When using D3D11\_BIND\_DEPTH\_STENCIL, MipLevels must be 1.
- When using D3D11\_BIND\_SHADER\_RESOURCE, SampleDesc.Count must be 1.
- When using D3D11\_BIND\_PRESENT, resource cannot have D3D11\_BIND\_SHADER\_RESOURCE.
- When using D3D10\_DDI\_RESOURCE\_MISC\_SHARED, Format cannot be DXGI\_FORMAT\_R8G8B8A8\_UNORM or DXGI\_FORMAT\_R8G8B8A8\_UNORM\_SRGB.

## ID3D11Device::CreateTexture3D

Feature Level	Maximum Dimension (any axis)	Dimensions must be power of two
D3D_FEATURE_LEVEL_9_1	256	Yes

Feature Level	Maximum Dimension (any axis)	Dimensions must be power of two
D3D_FEATURE_LEVEL_9_2	512	Yes
D3D_FEATURE_LEVEL_9_3	512	Yes

If resource is D3D11\_USAGE\_DEFAULT or D3D11\_USAGE\_IMMUTABLE, BindFlags cannot be zero.

## ID3D11Device::CreateUnorderedAccessView

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level. <del>{REMOVE}</del>
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11Device::CreateVertexShader

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Must use vs_4_0_level_9_1
D3D_FEATURE_LEVEL_9_2	Must use vs_4_0_level_9_1
D3D_FEATURE_LEVEL_9_3	Must use vs_4_0_level_9_3 or vs_4_0_level_9_1

## ID3D11Device::OpenSharedResource

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	<p>Use <a href="#">ID3D11Device::CheckFeatureSupport</a> with the <b>D3D11_FEATURE_FORMAT_SUPPORT2</b> value and the <b>D3D11_FEATURE_DATA_FORMAT_SUPPORT2</b> structure to determine if a format can be shared. If the format can be shared, <a href="#">CheckFeatureSupport</a> returns the <b>D3D11_FORMAT_SUPPORT2_SHAREABLE</b> flag.</p> <p>[!Note]  <b>DXGI_FORMAT_R8G8B8A8_UNORM</b> and <b>DXGI_FORMAT_R8G8B8A8_UNORM_SRGB</b> are never shareable when using feature level 9, even if the device indicates optional feature support for <b>D3D11_FORMAT_SUPPORT_SHAREABLE</b>. Attempting to create shared resources with DXGI formats <b>DXGI_FORMAT_R8G8B8A8_UNORM</b> and <b>DXGI_FORMAT_R8G8B8A8_UNORM_SRGB</b> will always fail unless the feature level is 10_0 or higher.</p> <p><del>{REMOVE}</del></p>

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## Related topics

[10Level9 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# 10Level9 ID3D11DeviceContext Methods

Article • 08/19/2021

This section lists the differences between each 10Level9 feature level and the D3D\_FEATURE\_LEVEL\_11\_0 and higher feature level for the [ID3D11DeviceContext](#) methods.

- [ID3D11DeviceContext::CopySubresourceRegion](#)
- [ID3D11DeviceContext::CopyResource](#)
- [ID3D11DeviceContext::CopyStructureCount](#)
- [ID3D11DeviceContext::ClearUnorderedAccessViewFloat](#)
- [ID3D11DeviceContext::ClearUnorderedAccessViewUint](#)
- [ID3D11DeviceContext::ClearRenderTargetView](#)
- [ID3D11DeviceContext::CSSetConstantBuffers](#)
- [ID3D11DeviceContext::CSSetSamplers](#)
- [ID3D11DeviceContext::CSSetShader](#)
- [ID3D11DeviceContext::CSSetShaderResources](#)
- [ID3D11DeviceContext::CSSetUnorderedAccessViews](#)
- [ID3D11DeviceContext::Dispatch](#)
- [ID3D11DeviceContext::DispatchIndirect](#)
- [ID3D11DeviceContext::Draw](#)
- [ID3D11DeviceContext::DrawAuto](#)
- [ID3D11DeviceContext::DrawIndexed](#)
- [ID3D11DeviceContext::DrawIndexedInstanced](#)
- [ID3D11DeviceContext::DrawIndexedInstancedIndirect](#)
- [ID3D11DeviceContext::DrawInstanced](#)
- [ID3D11DeviceContext::DrawInstancedIndirect](#)
- [ID3D11DeviceContext::DSSetConstantBuffers](#)
- [ID3D11DeviceContext::DSSetSamplers](#)
- [ID3D11DeviceContext::DSSetShader](#)
- [ID3D11DeviceContext::DSSetShaderResources](#)
- [ID3D11DeviceContext::GSSetConstantBuffers](#)
- [ID3D11DeviceContext::GSSetSamplers](#)
- [ID3D11DeviceContext::GSSetShader](#)
- [ID3D11DeviceContext::GSSetShaderResources](#)
- [ID3D11DeviceContext::HSSetConstantBuffers](#)
- [ID3D11DeviceContext::HSSetSamplers](#)
- [ID3D11DeviceContext::HSSetShader](#)
- [ID3D11DeviceContext::HSSetShaderResources](#)
- [ID3D11DeviceContext::IASetIndexBuffer](#)
- [ID3D11DeviceContext::IASetPrimitiveTopology](#)
- [ID3D11DeviceContext::OMSetBlendState](#)
- [ID3D11DeviceContext::OMSetRenderTargets](#)
- [ID3D11DeviceContext::OMSetRenderTargetAndUnorderedAccessViews](#)
- [ID3D11DeviceContext::PSSetConstantBuffers](#)
- [ID3D11DeviceContext::PSSetSamplers](#)
- [ID3D11DeviceContext::PSSetShader](#)
- [ID3D11DeviceContext::PSSetShaderResources](#)
- [ID3D11DeviceContext::RSSetScissorRects](#)
- [ID3D11DeviceContext::RSSetViewports](#)
- [ID3D11DeviceContext::SetPredication](#)
- [ID3D11DeviceContext::SOSetTargets](#)
- [ID3D11DeviceContext::VSSetConstantBuffers](#)
- [ID3D11DeviceContext::VSSetSamplers](#)
- [ID3D11DeviceContext::VSSetShader](#)

- [ID3D11DeviceContext::VSSetShaderResources](#)
- [Related topics](#)

## ID3D11DeviceContext::CopySubresourceRegion

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Only Texture2D and buffers may be copied within GPU-accessible memory. Texture3D cannot be copied from GPU-accessible memory to CPU-accessible memory. Any resource that has only D3D10_BIND_SHADER_RESOURCE cannot be copied from GPU-accessible memory to CPU-accessible memory. You can't copy mipmapped volume textures. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::CopyResource

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Only Texture2D and buffers may be copied within GPU-accessible memory. Texture3D cannot be copied from GPU-accessible memory to CPU-accessible memory. Any resource that has only D3D10_BIND_SHADER_RESOURCE cannot be copied from GPU-accessible memory to CPU-accessible memory. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::CopyStructureCount

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::ClearUnorderedAccessViewFloat

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::ClearUnorderedAccessViewUint

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::ClearRenderTargetView

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Only the first array slice will be cleared. Applications should create a render target view for each face or array slice, then clear each view individually.{REMOVE}
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::CSSetConstantBuffers

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.{REMOVE}
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::CSSetSamplers

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.{REMOVE}
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::CSSetShader

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.{REMOVE}
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::CSSetShaderResources

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.{REMOVE}
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::CSSetUnorderedAccessViews

Feature Level	Behavior Differences

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::Dispatch

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::DispatchIndirect

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::Draw

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Number of primitives may not exceed 65535. Textures cannot repeat over one primitive more than 128 times.
D3D_FEATURE_LEVEL_9_2	Number of primitives may not exceed 1048575. Textures cannot repeat over one primitive more than 2048 times.
D3D_FEATURE_LEVEL_9_3	Number of primitives may not exceed 1048575. Textures cannot repeat over one primitive more than 8192 times.

## ID3D11DeviceContext::DrawAuto

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::DrawIndexed

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Number of primitives may not exceed 65535. Textures cannot repeat over one primitive more than 128 times. Index values cannot exceed 65534. Indexed point lists not supported.

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_2	<p>Number of primitives may not exceed 1048575.</p> <p>Textures cannot repeat over one primitive more than 2048 times.</p> <p>Index values cannot exceed 1048575.</p> <p>Indexed point lists not supported.</p>
D3D_FEATURE_LEVEL_9_3	<p>Number of primitives may not exceed 1048575.</p> <p>Textures cannot repeat over one primitive more than 8192 times.</p> <p>Index values cannot exceed 1048575.</p> <p>Indexed point lists not supported.</p>

## ID3D11DeviceContext::DrawIndexedInstanced

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	<p>Number of primitives may not exceed 1048575.</p> <p>Textures cannot repeat over one primitive more than 8192 times.</p> <p>Index values cannot exceed 1048575.</p> <p>[!Note] When you call the <a href="#">DrawIndexedInstanced</a> method with a vertex shader that is bound to the pipeline and that doesn't import any per-instance data, some Direct3D 9 graphics hardware might not draw anything. In particular, if the vertex shader does not use any per-instance data, calling <a href="#">DrawIndexedInstanced</a> with 1 instance is not equivalent to calling <a href="#">Draw</a>.</p>

## ID3D11DeviceContext::DrawIndexedInstancedIndirect

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::DrawInstanced

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::DrawInstancedIndirect

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::DSSetConstantBuffers

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::DSSetSamplers

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::DSSetShader

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::DSSetShaderResources

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::GSSetConstantBuffers

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::GSSetSamplers

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::GSSetShader

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::GSSetShaderResources

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::HSSetConstantBuffers

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::HSSetSamplers

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::HSSetShader

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::HSSetShaderResources

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* or 10.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	
D3D_FEATURE_LEVEL_10_0	
D3D_FEATURE_LEVEL_10_1	

## ID3D11DeviceContext::IASetIndexBuffer

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Format is allowed to be different from that specified at buffer creation, but an expensive translation will be incurred. Only allows index buffers with the DXGI_FORMAT_R16_UINT format.
D3D_FEATURE_LEVEL_9_2	Format is allowed to be different from that specified at buffer creation, but an expensive translation will be incurred. Allows index buffers with the DXGI_FORMAT_R16_UINT and DXGI_FORMAT_R32_UINT formats like D3D_FEATURE_LEVEL_10_0 and higher. \${REMOVE}\$
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::IASetPrimitiveTopology

<b>Feature Level</b>	<b>Behavior Differences</b>
D3D_FEATURE_LEVEL_9_1	Primitive topologies with adjacency are not supported\${REMOVE}\$

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::OMSetBlendState

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	SampleMask cannot be zero\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::OMSetRenderTargets

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Only one render target supported\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	Only four render targets supported, and all bound resources must have same bit depth.

## ID3D11DeviceContext::OMSetRenderTargetsAndUnorderedAccessViews

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::PSSetConstantBuffers

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	See feature level 10.0, but total number of constants used by shader cannot exceed 32\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::PSSetSamplers

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	No more than 16 samplers can be bound\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::PSSetShader

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Only ps_4_0_level_9_1\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	Only ps_4_0_level_9_3 or ps_4_0_level_9_1

## ID3D11DeviceContext::PSSetShaderResources

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	No more than 8 simultaneously bound shader resources\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::RSSetScissorRects

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Only the zeroth scissor rect is available\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::RSSetViewports

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Only the zeroth viewport is available\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

Even though you specify float values to the members of the [D3D11\\_VIEWPORT](#) structure for the *pViewports* array in a call to [ID3D11DeviceContext::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.

The call to [ID3D11DeviceContext::RSSetViewports](#) for [feature levels](#) 10\_x and 11\_x works as you expect even when you use a negative top left corner for the viewport.

## ID3D11DeviceContext::SetPredication

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::SOSetTargets

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::VSSetConstantBuffers

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	See feature level 10.0, but total number of constants used by shader cannot exceed 255\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::VSSetSamplers

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## ID3D11DeviceContext::VSSetShader

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Only vs_4_0_level_9_1\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	Only vs_4_0_level_9_3 or vs_4_0_level_9_1

## ID3D11DeviceContext::VSSetShaderResources

Feature Level	Behavior Differences
D3D_FEATURE_LEVEL_9_1	Not supported on any 9.* feature level.\${REMOVE}\$
D3D_FEATURE_LEVEL_9_2	
D3D_FEATURE_LEVEL_9_3	

## Related topics

[10Level9 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Direct3D 11 Return Codes

Article • 08/19/2020

Return codes from API functions.

HRESULT	Description
D3D11_ERROR_FILE_NOT_FOUND (0x887C0002)	The file was not found.
D3D11_ERROR_TOO_MANY_UNIQUE_STATE_OBJECTS (0x887C0001)	There are too many unique instances of a particular type of state object.
D3D11_ERROR_TOO_MANY_UNIQUE_VIEW_OBJECTS (0x887C0003)	There are too many unique instances of a particular type of view object.
D3D11_ERROR_DEFERRED_CONTEXT_MAP_WITHOUT_INITIAL_DISCARD (0x887C0004)	The first call to <a href="#">ID3D11DeviceContext::Map</a> after either <a href="#">ID3D11Device::CreateDeferredContext</a> or <a href="#">ID3D11DeviceContext::FinishCommandList</a> per Resource was not D3D11_MAP_WRITE_DISCARD.
D3DERR_INVALIDCALL (replaced with DXGI_ERROR_INVALID_CALL) (0x887A0001)	The method call is invalid. For example, a method's parameter may not be a valid pointer.
D3DERR_WASSTILLDRAWING (replaced with DXGI_ERROR_WAS_STILL_DRAWING) (0x887A000A)	The previous blit operation that is transferring information to or from this surface is incomplete.
E_FAIL (0x80004005)	Attempted to create a device with the debug layer enabled and the layer is not installed.
E_INVALIDARG (0x80070057)	An invalid parameter was passed to the returning function.
E_OUTOFMEMORY (0x8007000E)	Direct3D could not allocate sufficient memory to complete the call.
E_NOTIMPL (0x80004001)	The method call isn't implemented with the passed parameter combination.
S_FALSE ((HRESULT)1L)	Alternate success value, indicating a successful but nonstandard completion (the precise meaning depends on context).
S_OK ((HRESULT)0L)	No error occurred.

For more return codes, see [DXGI\\_ERROR](#).

## Related topics

- Direct3D 11 Reference
- 

## Feedback

Was this page helpful? 👍 Yes 👎 No

[Get help at Microsoft Q&A](#)

# Common Version Reference

Article • 11/04/2020

The Direct3D API defines several API elements that are common to the Direct3D 12, Direct3D 11, Direct3D 10, and Direct3D 10.1. You can use these API elements in your code for any of these Direct3D versions. These API elements are known as version neutral.

## In this section

Topic	Description
<a href="#">Common Version Interfaces</a>	This section contains information about the common version interfaces.
<a href="#">Common Version Structures</a>	This section contains information about the common version structures.
<a href="#">Common Version Enumerations</a>	This section contains information about the common version enumerations.

## Related topics

[Direct3D 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Common version interfaces

Article • 10/06/2020

This section contains information about the common version interfaces.

## In this section

Topic	Description
<a href="#">ID3D10Blob</a>	This interface is used to return arbitrary-length data.
<a href="#">ID3DBlob</a>	This interface is used to return data of arbitrary length.
<a href="#">ID3DDestructionNotifier</a>	Used to register for callbacks when a Direct 3D nano-COM object is destroyed.
<a href="#">ID3DInclude</a>	<a href="#">ID3DInclude</a> is an include interface that the user implements to allow an application to call user-overridable methods for opening and closing shader #include files.
<a href="#">ID3DUserDefinedAnnotation</a>	The <a href="#">ID3DUserDefinedAnnotation</a> interface enables an application to describe conceptual sections and markers within the application's code flow. An appropriately enabled tool, such as Microsoft Visual Studio Ultimate 2012, can display these sections and markers visually along the tool's Microsoft Direct3D time line, while the tool debugs the application. These visual notes allow users of such a tool to navigate to parts of the time line that are of interest, or to understand what set of Direct3D calls are produced by certain sections of the application's code.

## Related topics

[Common version reference](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3D10Blob interface (d3dcommon.h)

Article02/22/2024

This interface is used to return arbitrary-length data.

## Inheritance

The **ID3D10Blob** interface inherits from the [IUnknown](#) interface. **ID3D10Blob** also has these types of members:

## Methods

The **ID3D10Blob** interface has these methods.

 Expand table

### [ID3D10Blob::GetBufferPointer](#)

Gets a pointer to the data.

### [ID3D10Blob::GetBufferSize](#)

Gets the size.

## Remarks

The [ID3DBlob](#) interface is type-defined in the D3DCCommon.h header file as a **ID3D10Blob** interface, which is fully defined in the D3DCCommon.h header file. **ID3DBlob** is version-neutral and can be used in code for any Direct3D version.

Blobs can be used as a data buffer, storing vertex, adjacency, and material information during mesh optimization and loading operations. Also, these objects are used to return object code and error messages in APIs that compile vertex, geometry and pixel shaders.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcommon.h

## See also

[Common Version Interfaces](#)

[ID3DBlob](#)

[IUnknown](#)

# ID3D10Blob::GetBufferPointer method (d3dcommon.h)

Article 02/22/2024

Gets a pointer to the data.

## Syntax

C++

```
LPVOID GetBufferPointer();
```

## Return value

Type: [LPVOID](#)

Returns a pointer.

## Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcommon.h

## See also

[ID3D10Blob](#)

# ID3D10Blob::GetBufferSize method (d3dcommon.h)

Article 02/22/2024

Gets the size.

## Syntax

C++

```
SIZE_T GetBufferSize();
```

## Return value

Type: [SIZE\\_T](#)

The size of the data, in bytes.

## Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcommon.h

## See also

[ID3D10Blob](#)

# ID3DBlob interface

Article 12/01/2017

This interface is used to return data of arbitrary length.

## Members

The **ID3DBlob** interface inherits from the [IUnknown](#) interface. **ID3DBlob** also has these types of members:

- Methods

## Methods

The **ID3DBlob** interface has these methods.

 [Expand table](#)

Method	Description
<a href="#">GetBufferPointer</a>	Retrieves a pointer to the blob's data.
<a href="#">GetBufferSize</a>	Retrieves the size, in bytes, of the blob's data.

## Remarks

An **ID3DBlob** is obtained by calling the [D3DCreateBlob](#) function. **D3DCreateBlob** is contained in D3dcompiler.lib or D3dcompiler\_nn.dll.

The **ID3DBlob** interface is type-defined in the D3DCCommon.h header file as a [ID3D10Blob](#) interface, which is fully defined in the D3DCCommon.h header file.

```
typedef ID3D10Blob ID3DBlob;
```

**ID3DBlob** is version-neutral and can be used in code for any Direct3D version.

Blobs can be used as data buffers. Blobs can also be used for storing vertex, adjacency, and material information during mesh optimization, and for loading operations. Also, these objects

are used to return object code and error messages in APIs that compile vertex, geometry, and pixel shaders.

## Requirements

 Expand table

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3Dcommon.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[Common Version Interfaces](#)

[D3DCreateBlob](#)

[ID3D10Blob in Direct3D 11](#)

[ID3D10Blob in Direct3D 10](#)

# ID3DBlob::GetBufferPointer method

Article 12/01/2017

Retrieves a pointer to the blob's data.

## Syntax

C++

```
LPVOID GetBufferPointer();
```

## Parameters

This method has no parameters.

## Return value

Type: [LPVOID](#)

Returns a pointer to the blob's data.

## Remarks

The [ID3DBlob](#) interface is type defined in the D3DCommon.h header file as a [ID3D10Blob](#) interface, which is fully defined in the D3DCommon.h header file.

```
typedef ID3D10Blob ID3DBlob;
```

[ID3DBlob](#) is version neutral and can be used in code for any Direct3D version.

## Requirements

 Expand table

Header	D3DCommon.h
Library	D3DCompiler.lib

## See also

[ID3DBlob](#)

# ID3DBlob::GetBufferSize method

Article 12/01/2017

Retrieves the size, in bytes, of the blob's data.

## Syntax

C++

```
SIZE_T GetBufferSize();
```

## Parameters

This method has no parameters.

## Return value

Type: [SIZE\\_T](#)

Returns the size of the blob's data, in bytes.

## Remarks

The [ID3DBlob](#) interface is type defined in the D3DCCommon.h header file as a [ID3D10Blob](#) interface, which is fully defined in the D3DCCommon.h header file.

```
typedef ID3D10Blob ID3DBlob;
```

[ID3DBlob](#) is version neutral and can be used in code for any Direct3D version.

## Requirements

 Expand table

Header	D3DCCommon.h
Library	D3DCompiler.lib

## See also

[ID3DBlob](#)

# ID3DDestructionNotifier interface (d3dcommon.h)

Article 02/10/2023

**ID3DDestructionNotifier** is an interface that you can use to register for callbacks when a Direct3D nano-COM object is destroyed.

To acquire an instance of this interface, call on a Direct3D object with the IID of **ID3DDestructionNotifier**.

Using **ID3DDestructionNotifier** instead of **ID3D12Object::SetPrivateDataInterface** or Direct3D 11 equivalents provides stronger guarantees about the order of destruction. With **ID3DDestructionNotifier**, implicit relationships—such as an **ID3D11View** holding a reference to its underlying **ID3D11Resource**—are guaranteed to be valid and for the referenced object (here, the **ID3D11Object**) to still be alive when the destruction callback is invoked. With **ID3D12Object::SetPrivateDataInterface**, the implicit references can be released before the destruction callback is invoked.

It isn't safe to access the object being destructed during the callback.

## Inheritance

The **ID3DDestructionNotifier** interface inherits from the **IUnknown** interface.

## Methods

The **ID3DDestructionNotifier** interface has these methods.

### [ID3DDestructionNotifier::RegisterDestructionCallback](#)

Registers a user-defined callback to be invoked on destruction of the object from which this **ID3DDestructionNotifier** was created.

### [ID3DDestructionNotifier::UnregisterDestructionCallback](#)

Unregisters a callback that was registered with [RegisterDestructionCallback](#).

## Remarks

The **ID3DDestructionNotifier** can be used to track resources which are being unexpectedly released early, or providing a log of object disposal.

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3dcommon.h

## See also

[ID3DDestructionNotifier::RegisterDestructionCallback](#)

[ID3DDestructionNotifier::UnregisterDestructionCallback](#)

[Common Version Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DDestructionNotifier::RegisterDestructionCallback method (d3dcommon.h)

Article 02/22/2024

Registers a user-defined callback to be invoked on destruction of the object from which this [ID3DDestructionNotifier](#) was created.

## Syntax

C++

```
HRESULT RegisterDestructionCallback(
    PFN_DESTRUCTION_CALLBACK callbackFn,
    void*                 pData,
    UINT                  *pCallbackID
);
```

## Parameters

`callbackFn`

Type: [PFN\\_DESTRUCTION\\_CALLBACK](#)

A user-defined callback to be invoked when the object is destroyed.

`pData`

Type: [void\\*](#)

The data to pass to *callbackFn* when invoked

`pCallbackID`

Type: [UINT\\*](#)

Pointer to a [UINT](#) used to identify the callback, and to pass to to unregister the callback.

## Return value

Type: [HRESULT](#)

If this function succeeds, it returns [S\\_OK](#).

# Remarks

An example of this interface being used to log the destruction of an **ID3D12Resource**.

C++/CX

```
#include <d3dcommon.h> // for ID3DDestructionNotifier

ComPtr<ID3D12Resource> resource = ...;

ComPtr<ID3DDestructionNotifier> notifier;
if (SUCCEEDED(resource.As(&notifier)))
{
    UINT callbackId;
    ThrowIfFailed(notifier->RegisterDestructionCallback(LogResourceReleased,
    nullptr, &callbackId));
}

void LogResourceReleased(void* context)
{
    OutputDebugString("Resource released!\n");
}
```

# Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcommon.h

# See also

[ID3DDestructionNotifier](#)

[ID3DDestructionNotifier::UnregisterDestructionCallback](#)

---

# Feedback

Was this page helpful?

 Yes

 No



# ID3DDestructionNotifier::UnregisterDestructionCallback method (d3dcommon.h)

Article 02/22/2024

Unregisters a callback that was registered with [RegisterDestructionCallback](#).

## Syntax

C++

```
HRESULT UnregisterDestructionCallback(  
    UINT callbackID  
) ;
```

## Parameters

`callbackID`

Type: [UINT](#)

The [UINT](#) that was created by the *pCallbackID* argument to [ID3DDestructionNotifier::RegisterDestructionCallback](#).

## Return value

Type: [HRESULT](#)

If this function succeeds, it returns [S\\_OK](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcommon.h

## See also

[ID3DDestructionNotifier](#)

[ID3DDestructionNotifier::RegisterDestructionCallback](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DInclude interface (d3dcommon.h)

Article 02/22/2024

**ID3DInclude** is an include interface that the user implements to allow an application to call user-overridable methods for opening and closing shader #include files.

## Methods

The **ID3DInclude** interface has these methods.

[+] Expand table

<b>ID3DInclude::Close</b>  A user-implemented method for closing a shader
<b>ID3DInclude::Open</b>  A user-implemented method for opening and reading the contents of a shader

## Remarks

To use this interface, create an interface that inherits from **ID3DInclude** and implement custom behavior for the methods.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3dcommon.h

## See also

[Common Version Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DInclude::Close method (d3dcommon.h)

Article 02/22/2024

A user-implemented method for closing a shader #include file.

## Syntax

C++

```
HRESULT Close(  
    LPCVOID pData  
);
```

## Parameters

pData

Type: [LPCVOID](#)

Pointer to the buffer that contains the include directives. This is the pointer that was returned by the corresponding [ID3DInclude::Open](#) call.

## Return value

Type: [HRESULT](#)

The user-implemented **Close** method should return **S\_OK**. If **Close** fails when it closes the #include file, the application programming interface (API) that caused **Close** to be called fails. This failure can occur in one of the following situations:

- The high-level shader language (HLSL) shader fails one of the [D3D10CompileShader\\*\\*\\*](#) functions.
- The effect fails one of the [D3D10CreateEffect\\*\\*\\*](#) functions.

## Remarks

If [ID3DInclude::Open](#) was successful, **Close** is guaranteed to be called before the API using the [ID3DInclude](#) interface returns.

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcommon.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3DInclude](#)

[ID3DInclude::Open](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DInclude::Open method (d3dcommon.h)

Article 02/22/2024

A user-implemented method for opening and reading the contents of a shader #include file.

## Syntax

C++

```
HRESULT Open(
    D3D_INCLUDE_TYPE IncludeType,
    LPCSTR          pFileName,
    LPCVOID         pParentData,
    LPCVOID         *ppData,
    UINT            *pBytes
);
```

## Parameters

IncludeType

Type: [D3D\\_INCLUDE\\_TYPE](#)

A [D3D\\_INCLUDE\\_TYPE](#)-typed value that indicates the location of the #include file.

pFileName

Type: [LPCSTR](#)

Name of the #include file.

pParentData

Type: [LPCVOID](#)

Pointer to the container that includes the #include file. The compiler might pass NULL in *pParentData*. For more information, see the "Searching for Include Files" section in [Compile an Effect \(Direct3D 11\)](#).

ppData

Type: [LPCVOID\\*](#)

Pointer to the buffer that contains the include directives. This pointer remains valid until you call [ID3DInclude::Close](#).

pBytes

Type: [UINT\\*](#)

Pointer to the number of bytes that [Open](#) returns in *ppData*.

## Return value

Type: [HRESULT](#)

The user-implemented method must return [S\\_OK](#). If [Open](#) fails when it reads the #include file, the application programming interface (API) that caused [Open](#) to be called fails. This failure can occur in one of the following situations:

- The high-level shader language (HLSL) shader fails one of the [D3D10CompileShader\\*\\*\\*](#) functions.
- The effect fails one of the [D3D10CreateEffect\\*\\*\\*](#) functions.

## Requirements

[\[ \]](#) Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcommon.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

## See also

[ID3DInclude](#)

[ID3DInclude::Close](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DUserDefinedAnnotation interface (d3d11\_1.h)

Article 07/22/2021

The **ID3DUserDefinedAnnotation** interface enables an application to describe conceptual sections and markers within the application's code flow. An appropriately enabled tool, such as Microsoft Visual Studio Ultimate 2012, can display these sections and markers visually along the tool's Microsoft Direct3D time line, while the tool debugs the application. These visual notes allow users of such a tool to navigate to parts of the time line that are of interest, or to understand what set of Direct3D calls are produced by certain sections of the application's code.

## Inheritance

The **ID3DUserDefinedAnnotation** interface inherits from the [IUnknown](#) interface. **ID3DUserDefinedAnnotation** also has these types of members:

## Methods

The **ID3DUserDefinedAnnotation** interface has these methods.

<a href="#">ID3DUserDefinedAnnotation::BeginEvent</a>
Marks the beginning of a section of event code.
<a href="#">ID3DUserDefinedAnnotation::EndEvent</a>
Marks the end of a section of event code.
<a href="#">ID3DUserDefinedAnnotation::GetStatus</a>
Determines whether the calling application is running under a Microsoft Direct3D profiling tool.
<a href="#">ID3DUserDefinedAnnotation::SetMarker</a>
Marks a single point of execution in code.

## Remarks

The methods of **ID3DUserDefinedAnnotation** have no effect when the calling application is not running under a Direct3D-specific profiling tool like Visual Studio Ultimate 2012.

The **ID3DUserDefinedAnnotation** interface is published by Microsoft Direct3D 11 device contexts. Therefore, **ID3DUserDefinedAnnotation** has the same threading rules as the **ID3D11DeviceContext** interface, or any other context interface. For more information about Direct3D threading, see [MultiThreading](#). To retrieve the **ID3DUserDefinedAnnotation** interface for the context, call the **QueryInterface** method for the context (for example, **ID3D11DeviceContext::QueryInterface**). In this call, you must pass the identifier of **ID3DUserDefinedAnnotation**.

The **ID3DUserDefinedAnnotation** interface is the Microsoft Direct3D 10 and later equivalent of the Direct3D 9 [PIX functions](#) (D3DPERF\_\* functions).

**Note** Setting the

**D3D11\_CREATE\_DEVICE\_PREVENT\_ALTERING\_LAYER\_SETTINGS\_FROM\_REGISTRY** flag in your app replaces calling **D3DPerf\_SetOptions(1)**. But, to prevent Direct3D debugging tools from hooking your app, your app can also call **ID3DUserDefinedAnnotation::GetStatus** to determine whether it is running under a Direct3D debugging tool and then exit accordingly.

You must call the **BeginEvent** and **EndEvent** methods in pairs; pairs of calls to these methods can nest within pairs of calls to these methods at a higher level in the application's call stack. In other words, a "Draw World" section can entirely contain another section named "Draw Trees," which can in turn entirely contain a section called "Draw Oaks." You can only associate an **EndEvent** method with the most recent **BeginEvent** method, that is, pairs cannot overlap. You cannot call an **EndEvent** for any **BeginEvent** that preceded the most recent **BeginEvent**. In fact, the runtime interprets the first **EndEvent** as ending the second **BeginEvent**.

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]

Target Platform	Windows
Header	d3d11_1.h

## See also

[Common Version Interfaces](#)

[IUnknown](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DUserDefinedAnnotation::BeginEvent method (d3d11\_1.h)

Article10/13/2021

Marks the beginning of a section of event code.

## Syntax

C++

```
INT BeginEvent(  
    [in] LPCWSTR Name  
);
```

## Parameters

[in] Name

A **NULL**-terminated **UNICODE** string that contains the name of the event. The name is not relevant to the operating system. You can choose a name that is meaningful when the calling application is running under the Direct3D profiling tool. A **NULL** pointer produces undefined results.

## Return value

Returns the number of previous calls to **BeginEvent** that have not yet been finalized by calls to the [ID3DUserDefinedAnnotation::EndEvent](#) method.

The return value is **-1** if the calling application is not running under a Direct3D profiling tool.

## Remarks

You call the [EndEvent](#) method to mark the end of the section of event code.

A user can visualize the event when the calling application is running under an enabled Direct3D profiling tool such as Microsoft Visual Studio Ultimate 2012.

**BeginEvent** has no effect if the calling application is not running under an enabled Direct3D profiling tool.

## Examples

The following code shows how to use a pair of calls to the **BeginEvent** and **EndEvent** methods. It also uses the [CComPtr](#) smart pointer type.

C++

```
CComPtr< ID3D11DeviceContext > pContext;

HRESULT hrCreateDevice = (*pfnD3D11CreateDevice)(
    0,
    D3D_DRIVER_TYPE_NULL,
    0,
    0,
    NULL,
    0,
    D3D11_SDK_VERSION,
    NULL,
    0,
    & pContext );
VERIFY_SUCCEEDED(hrCreateDevice);
CComPtr<ID3DUserDefinedAnnotation> pPerf;
HRESULT hr = pContext->QueryInterface( __uuidof(pPerf),
reinterpret_cast<void**>(&pPerf) );
if ( FAILED( hr ) )
    return;
pPerf->BeginEvent( L"Now entering ocean rendering code" );
MyDrawOceanRoutine( );
pPerf->EndEvent( );
```

## Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h

Library

D3D11.lib

## See also

[ID3DUserDefinedAnnotation](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DUserDefinedAnnotation::EndEvent method (d3d11\_1.h)

Article02/22/2024

Marks the end of a section of event code.

## Syntax

C++

```
INT EndEvent();
```

## Return value

Returns the number of previous calls to the [ID3DUserDefinedAnnotation::BeginEvent](#) method that have not yet been finalized by calls to [EndEvent](#).

The return value is  $-1$  if the calling application is not running under a Direct3D profiling tool.

## Remarks

You call the [BeginEvent](#) method to mark the beginning of the section of event code.

A user can visualize the event when the calling application is running under an enabled Direct3D profiling tool such as Microsoft Visual Studio Ultimate 2012.

[EndEvent](#) has no effect if the calling application is not running under an enabled Direct3D profiling tool.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3DUserDefinedAnnotation](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DUserDefinedAnnotation::GetStatus method (d3d11\_1.h)

Article02/22/2024

Determines whether the calling application is running under a Microsoft Direct3D profiling tool.

## Syntax

C++

```
BOOL GetStatus();
```

## Return value

The return value is nonzero if the calling application is running under a Direct3D profiling tool such as Visual Studio Ultimate 2012, and zero otherwise.

## Remarks

You can call **GetStatus** to determine whether your application is running under a Direct3D profiling tool before you make further calls to other methods of the [ID3DUserDefinedAnnotation](#) interface. For example, the [ID3DUserDefinedAnnotation::BeginEvent](#) and [ID3DUserDefinedAnnotation::EndEvent](#) methods have no effect if the calling application is not running under an enabled Direct3D profiling tool. Therefore, you do not need to call these methods unless your application is running under a Direct3D profiling tool.

## Examples

The following code shows how to use **GetStatus**.

```
#ifdef DEVELOPMENT_BUILD
    if ( pPerf->GetStatus() )
        m_MakeD3DAnnotationCalls = true;
#endif
```

...

```
if ( m_ MakeD3DAAnnotationCalls )
    pPerf->BeginEvent(L“Drawing Ocean”);
MyDrawOceanRoutine();
```

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3DUserDefinedAnnotation](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DUserDefinedAnnotation::SetMarker method (d3d11\_1.h)

Article 02/22/2024

Marks a single point of execution in code.

## Syntax

C++

```
void SetMarker(  
    [in] LPCWSTR Name  
);
```

## Parameters

[in] Name

A **N**ULL-terminated **UNICODE** string that contains the name of the marker. The name is not relevant to the operating system. You can choose a name that is meaningful when the calling application is running under the Direct3D profiling tool. A **N**ULL pointer produces undefined results.

## Return value

None

## Remarks

A user can visualize the marker when the calling application is running under an enabled Direct3D profiling tool such as Microsoft Visual Studio Ultimate 2012.

**SetMarker** has no effect if the calling application is not running under an enabled Direct3D profiling tool.

## Examples

The following code shows how to use **SetMarker**. It also uses the [CComPtr](#) smart pointer type.

```
CComPtr< ID3D11DeviceContext > pID3D11DeviceContext;

HRESULT hrCreateDevice = (*pfnD3D11CreateDevice)(
    0,
    D3D_DRIVER_TYPE_NULL,
    0,
    0,
    NULL,
    0,
    D3D11_SDK_VERSION,
    NULL,
    0,
    & pID3D11DeviceContext );
VERIFY_SUCCEEDED(hrCreateDevice);

CComPtr<ID3DUserDefinedAnnotation> pPerf;
HRESULT hr = pID3D11DeviceContext->QueryInterface( __uuidof(pPerf),
reinterpret_cast<void**>(&pPerf) );
if ( FAILED( hr ) )
    return;
pPerf->SetMarker( L"Occlusion test failed- not drawing sun flare" );
```

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11_1.h
Library	D3D11.lib

## See also

[ID3DUserDefinedAnnotation](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# Common Version Structures

Article • 11/04/2020

This section contains information about the common version structures.

## In this section

Topic	Description
<a href="#">D3D_SHADER_MACRO</a>	Defines a shader macro.

## Related topics

[Common Version Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D\_SHADER\_MACRO structure (d3dcommon.h)

Article 02/22/2024

Defines a shader macro.

## Syntax

C++

```
typedef struct _D3D_SHADER_MACRO {
    LPCSTR Name;
    LPCSTR Definition;
} D3D_SHADER_MACRO, *LPD3D_SHADER_MACRO;
```

## Members

Name

The macro name.

Definition

The macro definition.

## Remarks

You can use shader macros in your shaders. The **D3D\_SHADER\_MACRO** structure defines a single shader macro as shown in the following example:

```
D3D_SHADER_MACRO Shader_Macros[] = { "zero", "0", NULL, NULL };
```

The following shader or effect creation functions take an array of shader macros as an input parameter:

- [D3D10CompileShader](#)

- D3DX10CreateEffectFromFile
- D3DX10PreprocessShaderFromFile
- D3DX11CreateAsyncShaderPreprocessProcessor

## Requirements

 Expand table

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# Common Version Enumerations

Article • 11/04/2020

This section contains information about the common version enumerations.

## In this section

Topic	Description
<a href="#">D3D_CBUFFER_TYPE</a>	Values that identify the intended use of constant-buffer data.
<a href="#">D3D_DRIVER_TYPE</a>	Driver type options.
<a href="#">D3D_FEATURE_LEVEL</a>	Describes the set of features targeted by a Direct3D device.
<a href="#">D3D_INCLUDE_TYPE</a>	Values that indicate the location of a shader #include file.
<a href="#">D3D_INTERPOLATION_MODE</a>	Specifies interpolation mode, which affects how values are calculated during rasterization.
<a href="#">D3D_MIN_PRECISION</a>	Values that indicate the minimum desired interpolation precision.
<a href="#">D3D_NAME</a>	Values that identify shader parameters that use system-value semantics.
<a href="#">D3D_PRIMITIVE</a>	Values that indicate how the pipeline interprets geometry or hull shader input primitives.
<a href="#">D3D_PRIMITIVE_TOPOLOGY</a>	Values that indicate how the pipeline interprets vertex data that is bound to the <a href="#">input-assembler stage</a> . These <a href="#">primitive topology values</a> determine how the vertex data is rendered on screen.
<a href="#">D3D_REGISTER_COMPONENT_TYPE</a>	Values that identify the data types that can be stored in a register.
<a href="#">D3D_RESOURCE_RETURN_TYPE</a>	Values that identify the return type of a resource.
<a href="#">D3D_SHADER_CBUFFER_FLAGS</a>	Values that identify the intended use of a constant-data buffer.
<a href="#">D3D_SHADER_INPUT_FLAGS</a>	Values that identify shader-input options.

Topic	Description
<a href="#">D3D_SHADER_INPUT_TYPE</a>	Values that identify resource types that can be bound to a shader and that are reflected as part of the resource description for the shader.
<a href="#">D3D_SHADER_VARIABLE_CLASS</a>	Values that identify the class of a shader variable.
<a href="#">D3D_SHADER_VARIABLE_FLAGS</a>	Values that identify information about a shader variable.
<a href="#">D3D_SHADER_VARIABLE_TYPE</a>	Values that identify various data, texture, and buffer types that can be assigned to a shader variable.
<a href="#">D3D_SRV_DIMENSION</a>	Values that identify the type of resource to be viewed as a shader resource.
<a href="#">D3D_TESSELLATOR_DOMAIN</a>	Values that identify domain options for tessellator data.
<a href="#">D3D_TESSELLATOR_OUTPUT_PRIMITIVE</a>	Values that identify output primitive types.
<a href="#">D3D_TESSELLATOR_PARTITIONING</a>	Values that identify partitioning options.

## Related topics

[Common Version Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D\_CBUFFER\_TYPE enumeration (d3dcommon.h)

Article 02/22/2024

Values that identify the intended use of constant-buffer data.

## ⓘ Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum _D3D_CBUFFER_TYPE {
    D3D_CT_CBUFFER = 0,
    D3D_CT_TBUFFER,
    D3D_CT_INTERFACE_POINTERS,
    D3D_CT_RESOURCE_BIND_INFO,
    D3D10_CT_CBUFFER,
    D3D10_CT_TBUFFER,
    D3D11_CT_CBUFFER,
    D3D11_CT_TBUFFER,
    D3D11_CT_INTERFACE_POINTERS,
    D3D11_CT_RESOURCE_BIND_INFO
} D3D_CBUFFER_TYPE;
```

## Constants

[ ] Expand table

`D3D_CT_CBUFFER`

Value: *0*

A buffer containing scalar constants.

`D3D_CT_TBUFFER`

A buffer containing texture data.

#### D3D\_CT\_INTERFACE\_POINTERS

A buffer containing interface pointers.

#### D3D\_CT\_RESOURCE\_BIND\_INFO

A buffer containing binding information.

#### D3D10\_CT\_CBUFFER

A buffer containing scalar constants.

#### D3D10\_CT\_TBUFFER

A buffer containing texture data.

#### D3D11\_CT\_CBUFFER

A buffer containing scalar constants.

#### D3D11\_CT\_TBUFFER

A buffer containing texture data.

#### D3D11\_CT\_INTERFACE\_POINTERS

A buffer containing interface pointers.

#### D3D11\_CT\_RESOURCE\_BIND\_INFO

A buffer containing binding information.

## Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_DRIVER\_TYPE enumeration (d3dcommon.h)

Article 02/22/2024

Driver type options.

## ⓘ Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_DRIVER_TYPE {
    D3D_DRIVER_TYPE_UNKNOWN = 0,
    D3D_DRIVER_TYPE_HARDWARE,
    D3D_DRIVER_TYPE_REFERENCE,
    D3D_DRIVER_TYPE_NULL,
    D3D_DRIVER_TYPE_SOFTWARE,
    D3D_DRIVER_TYPE_WARP
} ;
```

## Constants

[+] Expand table

`D3D_DRIVER_TYPE_UNKNOWN`

Value: 0

The driver type is unknown.

`D3D_DRIVER_TYPE_HARDWARE`

A hardware driver, which implements Direct3D features in hardware. This is the primary driver that you should use in your Direct3D applications because it provides the best performance. A hardware driver uses hardware acceleration (on supported hardware) but can also use software for parts of the pipeline that are not supported in hardware. This driver type is often referred to as a hardware abstraction layer or HAL.

#### D3D\_DRIVER\_TYPE\_REFERENCE

A reference driver, which is a software implementation that supports every Direct3D feature. A reference driver is designed for accuracy rather than speed and as a result is slow but accurate. The rasterizer portion of the driver does make use of special CPU instructions whenever it can, but it is not intended for retail applications; use it only for feature testing, demonstration of functionality, debugging, or verifying bugs in other drivers. The reference device for this driver is installed by the Windows SDK 8.0 or later and is intended only as a debug aid for development purposes. This driver may be referred to as a REF driver, a reference driver, or a reference rasterizer.

**Note** When you use the REF driver in Windows Store apps, the REF driver renders correctly but doesn't display any output on the screen. To verify bugs in hardware drivers for Windows Store apps, use [D3D DRIVER TYPE WARP](#) for the WARP driver instead.

#### D3D\_DRIVER\_TYPE\_NULL

A NULL driver, which is a reference driver without render capability. This driver is commonly used for debugging non-rendering API calls, it is not appropriate for retail applications. This driver is installed by the DirectX SDK.

#### D3D\_DRIVER\_TYPE\_SOFTWARE

A software driver, which is a driver implemented completely in software. The software implementation is not intended for a high-performance application due to its very slow performance.

#### D3D\_DRIVER\_TYPE\_WARP

A WARP driver, which is a high-performance software rasterizer. The rasterizer supports [feature levels](#) 9\_1 through level 10\_1 with a high performance software implementation. For information about limitations creating a WARP device on certain feature levels, see [Limitations Creating WARP and Reference Devices](#). For more information about using a WARP driver, see [Windows Advanced Rasterization Platform \(WARP\) In-Depth Guide](#).

**Note** The WARP driver that Windows 8 includes supports [feature levels](#) 9\_1 through level 11\_1.

**Note** The WARP driver that Windows 8.1 includes fully supports [feature level 11\\_1](#), including tiled resources, [IDXGIDevice3::Trim](#), shared BCn surfaces, minblend, and map default.

## Remarks

The driver type is required when calling [D3D11CreateDevice](#) or [D3D11CreateDeviceAndSwapChain](#).

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# D3D\_FEATURE\_LEVEL enumeration (d3dcommon.h)

Article 02/22/2024

Describes the set of features targeted by a Direct3D device.

## Syntax

C++

```
typedef enum D3D_FEATURE_LEVEL {
    D3D_FEATURE_LEVEL_1_0_GENERIC,
    D3D_FEATURE_LEVEL_1_0_CORE,
    D3D_FEATURE_LEVEL_9_1,
    D3D_FEATURE_LEVEL_9_2,
    D3D_FEATURE_LEVEL_9_3,
    D3D_FEATURE_LEVEL_10_0,
    D3D_FEATURE_LEVEL_10_1,
    D3D_FEATURE_LEVEL_11_0,
    D3D_FEATURE_LEVEL_11_1,
    D3D_FEATURE_LEVEL_12_0,
    D3D_FEATURE_LEVEL_12_1,
    D3D_FEATURE_LEVEL_12_2
} ;
```

## Constants

[+] Expand table

`D3D_FEATURE_LEVEL_1_0_CORE`

Value: `(0x1000)`

Allows Microsoft Compute Driver Model (MCDM) devices to be used, or more feature-rich devices (such as traditional GPUs) that support a superset of the functionality. MCDM is the overall driver model for compute-only; it's a scaled-down peer of the larger scoped Windows Device Driver Model (WDDM).

`D3D_FEATURE_LEVEL_9_1`

Value: `(0x9100)`

Targets features supported by **feature level** 9.1, including shader model 2.

`D3D_FEATURE_LEVEL_9_2`

Value: `(0x9200)`

Targets features supported by <a href="#">feature level</a> 9.2, including shader model 2.
<b>D3D_FEATURE_LEVEL_9_3</b> Value: (0x9300) Targets features supported by <a href="#">feature level</a> 9.3, including shader model 2.0b.
<b>D3D_FEATURE_LEVEL_10_0</b> Value: (0xa000) Targets features supported by Direct3D 10.0, including shader model 4.
<b>D3D_FEATURE_LEVEL_10_1</b> Value: (0xa100) Targets features supported by Direct3D 10.1, including shader model 4.
<b>D3D_FEATURE_LEVEL_11_0</b> Value: (0xb000) Targets features supported by Direct3D 11.0, including shader model 5.
<b>D3D_FEATURE_LEVEL_11_1</b> Value: (0xb100) Targets features supported by Direct3D 11.1, including shader model 5 and logical blend operations. This feature level requires a display driver that is at least implemented to WDDM for Windows 8 (WDDM 1.2).
<b>D3D_FEATURE_LEVEL_12_0</b> Value: (0xc000) Targets features supported by Direct3D 12.0, including shader model 5.
<b>D3D_FEATURE_LEVEL_12_1</b> Value: (0xc100) Targets features supported by Direct3D 12.1, including shader model 5.
<b>D3D_FEATURE_LEVEL_12_2</b> Value: (0xc200) Targets features supported by Direct3D 12.2, including shader model 6.5. For more information about feature level 12_2, see its <a href="#">specification page</a> . Feature level 12_2 is available in Windows SDK builds 20170 and later.

## Remarks

For an overview of the capabilities of each feature level, see [Direct3D feature levels](#).

For information about limitations creating non-hardware-type devices on certain feature levels, see [Limitations creating WARP and reference devices](#).

## Requirements

Requirement	Value
Header	d3dcommon.h

## See also

- [Common version enumerations](#)

---

## Feedback

Was this page helpful?

[!\[\]\(4b02c3185eb8f437fd289f4413a15079\_img.jpg\) Yes](#)[!\[\]\(590beab0e9e0716c3d939529fca6246d\_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_INCLUDE\_TYPE enumeration (d3dcommon.h)

Article 02/22/2024

Values that indicate the location of a shader #include file.

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum _D3D_INCLUDE_TYPE {
    D3D_INCLUDE_LOCAL = 0,
    D3D_INCLUDE_SYSTEM,
    D3D10_INCLUDE_LOCAL,
    D3D10_INCLUDE_SYSTEM,
    D3D_INCLUDE_FORCE_DWORD = 0x7fffffff
} D3D_INCLUDE_TYPE;
```

## Constants

[ ] Expand table

`D3D_INCLUDE_LOCAL`

Value: 0

The local directory.

`D3D_INCLUDE_SYSTEM`

The system directory.

`D3D10_INCLUDE_LOCAL`

The local directory.

`D3D10_INCLUDE_SYSTEM`

The system directory.

D3D\_INCLUDE\_FORCE\_DWORD

Value: 0x7fffffff

Forces this enumeration to compile to 32 bits in size. Without this value, some compilers would allow this enumeration to compile to a size other than 32 bits.

Do not use this value.

## Remarks

You pass a D3D\_INCLUDE\_TYPE-typed value to the *IncludeType* parameter in a call to the [ID3DInclude::Open](#) method to indicate the location of the #include file.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

[D3D\\_INCLUDE\\_TYPE](#)

[ID3DInclude::Open](#)

# D3D\_INTERPOLATION\_MODE enumeration (d3dcommon.h)

Article 02/22/2024

Specifies interpolation mode, which affects how values are calculated during rasterization.

## Syntax

C++

```
typedef enum D3D_INTERPOLATION_MODE {
    D3D_INTERPOLATION_UNDEFINED = 0,
    D3D_INTERPOLATION_CONSTANT = 1,
    D3D_INTERPOLATION_LINEAR = 2,
    D3D_INTERPOLATION_LINEAR_CENTROID = 3,
    D3D_INTERPOLATION_LINEAR_NOPERSPECTIVE = 4,
    D3D_INTERPOLATION_LINEAR_NOPERSPECTIVE_CENTROID = 5,
    D3D_INTERPOLATION_LINEAR_SAMPLE = 6,
    D3D_INTERPOLATION_LINEAR_NOPERSPECTIVE_SAMPLE = 7
} ;
```

## Constants

[] Expand table

**D3D\_INTERPOLATION\_UNDEFINED**

Value: 0

The interpolation mode is undefined.

**D3D\_INTERPOLATION\_CONSTANT**

Value: 1

Don't interpolate between register values.

**D3D\_INTERPOLATION\_LINEAR**

Value: 2

Interpolate linearly between register values.

**D3D\_INTERPOLATION\_LINEAR\_CENTROID**

Value: 3

Interpolate linearly between register values but centroid clamped when multisampling.

`D3D_INTERPOLATION_LINEAR_NOPERSPECTIVE`

Value: 4

Interpolate linearly between register values but with no perspective correction.

`D3D_INTERPOLATION_LINEAR_NOPERSPECTIVE_CENTROID`

Value: 5

Interpolate linearly between register values but with no perspective correction and centroid clamped when multisampling.

`D3D_INTERPOLATION_LINEAR_SAMPLE`

Value: 6

Interpolate linearly between register values but sample clamped when multisampling.

`D3D_INTERPOLATION_LINEAR_NOPERSPECTIVE_SAMPLE`

Value: 7

Interpolate linearly between register values but with no perspective correction and sample clamped when multisampling.

## Requirements

[ ] Expand table

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

[D3D11\\_PARAMETER\\_DESC](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_MIN\_PRECISION enumeration (d3dcommon.h)

Article 02/22/2024

Values that indicate the minimum desired interpolation precision.

## Syntax

C++

```
typedef enum D3D_MIN_PRECISION {
    D3D_MIN_PRECISION_DEFAULT = 0,
    D3D_MIN_PRECISION_FLOAT_16 = 1,
    D3D_MIN_PRECISION_FLOAT_2_8 = 2,
    D3D_MIN_PRECISION_RESERVED = 3,
    D3D_MIN_PRECISION_SINT_16 = 4,
    D3D_MIN_PRECISION_UINT_16 = 5,
    D3D_MIN_PRECISION_ANY_16 = 0xf0,
    D3D_MIN_PRECISION_ANY_10 = 0xf1
} ;
```

## Constants

[+] Expand table

	<b>D3D_MIN_PRECISION_DEFAULT</b>
Value:	0
	Default minimum precision, which is 32-bit precision.
	<b>D3D_MIN_PRECISION_FLOAT_16</b>
Value:	1
	Minimum precision is min16float, which is 16-bit floating point.
	<b>D3D_MIN_PRECISION_FLOAT_2_8</b>
Value:	2
	Minimum precision is min10float, which is 10-bit floating point.
	<b>D3D_MIN_PRECISION_RESERVED</b>
Value:	3
	Reserved

#### D3D\_MIN\_PRECISION\_SINT\_16

Value: 4

Minimum precision is min16int, which is 16-bit signed integer.

#### D3D\_MIN\_PRECISION\_UINT\_16

Value: 5

Minimum precision is min16uint, which is 16-bit unsigned integer.

#### D3D\_MIN\_PRECISION\_ANY\_16

Value: 0xf0

Minimum precision is any 16-bit value.

#### D3D\_MIN\_PRECISION\_ANY\_10

Value: 0xf1

Minimum precision is any 10-bit value.

## Remarks

For more info, see [Scalar Types](#) and [Using HLSL minimum precision](#).

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# D3D\_NAME enumeration (d3dcommon.h)

Article01/31/2022

Values that identify shader parameters that use system-value semantics.

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_NAME {
    D3D_NAME_UNDEFINED = 0,
    D3D_NAME_POSITION = 1,
    D3D_NAME_CLIP_DISTANCE = 2,
    D3D_NAME_CULL_DISTANCE = 3,
    D3D_NAME_RENDER_TARGET_ARRAY_INDEX = 4,
    D3D_NAME_VIEWPORT_ARRAY_INDEX = 5,
    D3D_NAME_VERTEX_ID = 6,
    D3D_NAME_PRIMITIVE_ID = 7,
    D3D_NAME_INSTANCE_ID = 8,
    D3D_NAME_IS_FRONT_FACE = 9,
    D3D_NAME_SAMPLE_INDEX = 10,
    D3D_NAME_FINAL_QUAD_EDGE_TESSFACTOR = 11,
    D3D_NAME_FINAL_QUAD_INSIDE_TESSFACTOR = 12,
    D3D_NAME_FINAL_TRI_EDGE_TESSFACTOR = 13,
    D3D_NAME_FINAL_TRI_INSIDE_TESSFACTOR = 14,
    D3D_NAME_FINAL_LINE_DETAIL_TESSFACTOR = 15,
    D3D_NAME_FINAL_LINE_DENSITY_TESSFACTOR = 16,
    D3D_NAME_BARYCENTRICS = 23,
    D3D_NAME_SHADINGRATE,
    D3D_NAME_CULLPRIMITIVE,
    D3D_NAME_TARGET = 64,
    D3D_NAME_DEPTH = 65,
    D3D_NAME_COVERAGE = 66,
    D3D_NAME_DEPTH_GREATER_EQUAL = 67,
    D3D_NAME_DEPTH_LESS_EQUAL = 68,
    D3D_NAME_STENCIL_REF = 69,
    D3D_NAME_INNER_COVERAGE = 70,
    D3D10_NAME_UNDEFINED,
    D3D10_NAME_POSITION,
    D3D10_NAME_CLIP_DISTANCE,
    D3D10_NAME_CULL_DISTANCE,
    D3D10_NAME_RENDER_TARGET_ARRAY_INDEX,
```

```
D3D10_NAME_VIEWPORT_ARRAY_INDEX,  
D3D10_NAME_VERTEX_ID,  
D3D10_NAME_PRIMITIVE_ID,  
D3D10_NAME_INSTANCE_ID,  
D3D10_NAME_IS_FRONT_FACE,  
D3D10_NAME_SAMPLE_INDEX,  
D3D10_NAME_TARGET,  
D3D10_NAME_DEPTH,  
D3D10_NAME_COVERAGE,  
D3D11_NAME_FINAL_QUAD_EDGE_TESSFACTOR,  
D3D11_NAME_FINAL_QUAD_INSIDE_TESSFACTOR,  
D3D11_NAME_FINAL_TRI_EDGE_TESSFACTOR,  
D3D11_NAME_FINAL_TRI_INSIDE_TESSFACTOR,  
D3D11_NAME_FINAL_LINE_DETAIL_TESSFACTOR,  
D3D11_NAME_FINAL_LINE_DENSITY_TESSFACTOR,  
D3D11_NAME_DEPTH_GREATER_EQUAL,  
D3D11_NAME_DEPTH_LESS_EQUAL,  
D3D11_NAME_STENCIL_REF,  
D3D11_NAME_INNER_COVERAGE,  
D3D12_NAME_BARYCENTRICS,  
D3D12_NAME_SHADINGRATE,  
D3D12_NAME_CULLPRIMITIVE  
} ;
```

## Constants

  [Expand table](#)

**D3D\_NAME\_UNDEFINED**

Value: 0

This parameter does not use a predefined system-value semantic.

**D3D\_NAME\_POSITION**

Value: 1

This parameter contains position data.

**D3D\_NAME\_CLIP\_DISTANCE**

Value: 2

This parameter contains clip-distance data.

**D3D\_NAME\_CULL\_DISTANCE**

Value: 3

This parameter contains cull-distance data.

**D3D\_NAME\_RENDER\_TARGET\_ARRAY\_INDEX**

Value: 4

This parameter contains a render-target-array index.

`D3D_NAME_VIEWPORT_ARRAY_INDEX`

Value: 5

This parameter contains a viewport-array index.

`D3D_NAME_VERTEX_ID`

Value: 6

This parameter contains a vertex ID.

`D3D_NAME_PRIMITIVE_ID`

Value: 7

This parameter contains a primitive ID.

`D3D_NAME_INSTANCE_ID`

Value: 8

This parameter contains an instance ID.

`D3D_NAME_IS_FRONT_FACE`

Value: 9

This parameter contains data that identifies whether or not the primitive faces the camera.

`D3D_NAME_SAMPLE_INDEX`

Value: 10

This parameter contains a sampler-array index.

`D3D_NAME_FINAL_QUAD_EDGE_TESSFACTOR`

Value: 11

This parameter contains one of four tessellation factors that correspond to the amount of parts that a quad patch is broken into along the given edge. This flag is used to tessellate a quad patch.

`D3D_NAME_FINAL_QUAD_INSIDE_TESSFACTOR`

Value: 12

This parameter contains one of two tessellation factors that correspond to the amount of parts that a quad patch is broken into vertically and horizontally within the patch. This flag is used to tessellate a quad patch.

`D3D_NAME_FINAL_TRI_EDGE_TESSFACTOR`

Value: 13

This parameter contains one of three tessellation factors that correspond to the amount of parts that a tri patch is broken into along the given edge. This flag is used to tessellate a tri patch.

`D3D_NAME_FINAL_TRI_INSIDE_TESSFACTOR`

Value: 14

This parameter contains the tessellation factor that corresponds to the amount of parts that a tri patch is broken into within the patch. This flag is used to tessellate a tri patch.

`D3D_NAME_FINAL_LINE_DETAIL_TESSFACTOR`

Value: 15

This parameter contains the tessellation factor that corresponds to the number of lines broken into within the patch. This flag is used to tessellate an isolines patch.

`D3D_NAME_FINAL_LINE_DENSITY_TESSFACTOR`

Value: 16

This parameter contains the tessellation factor that corresponds to the number of lines that are created within the patch. This flag is used to tessellate an isolines patch.

`D3D_NAME_BARYCENTRICS`

Value: 23

This parameter contains barycentric coordinate data.

`D3D_NAME_TARGET`

Value: 64

This parameter contains render-target data.

`D3D_NAME_DEPTH`

Value: 65

This parameter contains depth data.

`D3D_NAME_COVERAGE`

Value: 66

This parameter contains alpha-coverage data.

`D3D_NAME_DEPTH_GREATER_EQUAL`

Value: 67

This parameter signifies that the value is greater than or equal to a reference value. This flag is used to specify conservative depth for a pixel shader.

`D3D_NAME_DEPTH_LESS_EQUAL`

Value: 68

This parameter signifies that the value is less than or equal to a reference value. This flag is used to specify conservative depth for a pixel shader.

`D3D_NAME_STENCIL_REF`

Value: 69

This parameter contains a stencil reference.

See [Shader Specified Stencil Reference Value](#).

`D3D_NAME_INNER_COVERAGE`

Value: 70

This parameter contains inner input coverage data.

See [Conservative Rasterization](#).

`D3D10_NAME_UNDEFINED`

This parameter does not use a predefined system-value semantic.

`D3D10_NAME_POSITION`

This parameter contains position data.

**D3D10\_NAME\_CLIP\_DISTANCE**

This parameter contains clip-distance data.

**D3D10\_NAME\_CULL\_DISTANCE**

This parameter contains cull-distance data.

**D3D10\_NAME\_RENDER\_TARGET\_ARRAY\_INDEX**

This parameter contains a render-target-array index.

**D3D10\_NAME\_VIEWPORT\_ARRAY\_INDEX**

This parameter contains a viewport-array index.

**D3D10\_NAME\_VERTEX\_ID**

This parameter contains a vertex ID.

**D3D10\_NAME\_PRIMITIVE\_ID**

This parameter contains a primitive ID.

**D3D10\_NAME\_INSTANCE\_ID**

This parameter contains a instance ID.

**D3D10\_NAME\_IS\_FRONT\_FACE**

This parameter contains data that identifies whether or not the primitive faces the camera.

**D3D10\_NAME\_SAMPLE\_INDEX**

This parameter contains a sampler-array index.

**D3D10\_NAME\_TARGET**

This parameter contains render-target data.

**D3D10\_NAME\_DEPTH**

This parameter contains depth data.

**D3D10\_NAME\_COVERAGE**

This parameter contains alpha-coverage data.

**D3D11\_NAME\_FINAL\_QUAD\_EDGE\_TESSFACTOR**

This parameter contains one of four tessellation factors that correspond to the amount of parts that a quad patch is broken into along the given edge. This flag is used to tessellate a quad patch.

**D3D11\_NAME\_FINAL\_QUAD\_INSIDE\_TESSFACTOR**

This parameter contains one of two tessellation factors that correspond to the amount of parts that a quad patch is broken into vertically and horizontally within the patch. This flag is used to tessellate a quad patch.

**D3D11\_NAME\_FINAL\_TRI\_EDGE\_TESSFACTOR**

This parameter contains one of three tessellation factors that correspond to the amount of parts that a tri patch is broken into along the given edge. This flag is used to tessellate a tri patch.

**D3D11\_NAME\_FINAL\_TRI\_INSIDE\_TESSFACTOR**

This parameter contains the tessellation factor that corresponds to the amount of parts that a tri patch is broken into within the patch. This flag is used to tessellate a tri patch.

**D3D11\_NAME\_FINAL\_LINE\_DETAIL\_TESSFACTOR**

This parameter contains the tessellation factor that corresponds to the amount of lines broken into within the patch. This flag is used to tessellate an isolines patch.

**D3D11\_NAME\_FINAL\_LINE\_DENSITY\_TESSFACTOR**

This parameter contains the tessellation factor that corresponds to the amount of lines that are created within the patch. This flag is used to tessellate an isolines patch.

**D3D11\_NAME\_DEPTH\_GREATER\_EQUAL**

This parameter signifies that the value is greater than or equal to a reference value. This flag is used to specify conservative depth for a pixel shader.

**D3D11\_NAME\_DEPTH\_LESS\_EQUAL**

This parameter signifies that the value is less than or equal to a reference value. This flag is used to specify conservative depth for a pixel shader.

**D3D11\_NAME\_STENCIL\_REF**

This parameter contains a stencil reference.

See [Shader Specified Stencil Reference Value](#).

**D3D11\_NAME\_INNER\_COVERAGE**

This parameter contains inner input coverage data.

See [Conservative Rasterization](#).

**D3D12\_NAME\_BARYCENTRICS**

This parameter contains barycentric coordinate data.

## Remarks

The **D3D\_NAME** values identify shader parameters that have [predefined system-value](#) semantics. These values are used in a shader-signature description. For more information about shader-signature description, see [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#).

## Requirements

 [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

[D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#)

# D3D\_PRIMITIVE enumeration (d3dcommon.h)

Article01/31/2022

Indicates how the pipeline interprets geometry or hull shader input primitives.

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_PRIMITIVE {
    D3D_PRIMITIVE_UNDEFINED = 0,
    D3D_PRIMITIVE_POINT = 1,
    D3D_PRIMITIVE_LINE = 2,
    D3D_PRIMITIVE_TRIANGLE = 3,
    D3D_PRIMITIVE_LINE_ADJ = 6,
    D3D_PRIMITIVE_TRIANGLE_ADJ = 7,
    D3D_PRIMITIVE_1_CONTROL_POINT_PATCH = 8,
    D3D_PRIMITIVE_2_CONTROL_POINT_PATCH = 9,
    D3D_PRIMITIVE_3_CONTROL_POINT_PATCH = 10,
    D3D_PRIMITIVE_4_CONTROL_POINT_PATCH = 11,
    D3D_PRIMITIVE_5_CONTROL_POINT_PATCH = 12,
    D3D_PRIMITIVE_6_CONTROL_POINT_PATCH = 13,
    D3D_PRIMITIVE_7_CONTROL_POINT_PATCH = 14,
    D3D_PRIMITIVE_8_CONTROL_POINT_PATCH = 15,
    D3D_PRIMITIVE_9_CONTROL_POINT_PATCH = 16,
    D3D_PRIMITIVE_10_CONTROL_POINT_PATCH = 17,
    D3D_PRIMITIVE_11_CONTROL_POINT_PATCH = 18,
    D3D_PRIMITIVE_12_CONTROL_POINT_PATCH = 19,
    D3D_PRIMITIVE_13_CONTROL_POINT_PATCH = 20,
    D3D_PRIMITIVE_14_CONTROL_POINT_PATCH = 21,
    D3D_PRIMITIVE_15_CONTROL_POINT_PATCH = 22,
    D3D_PRIMITIVE_16_CONTROL_POINT_PATCH = 23,
    D3D_PRIMITIVE_17_CONTROL_POINT_PATCH = 24,
    D3D_PRIMITIVE_18_CONTROL_POINT_PATCH = 25,
    D3D_PRIMITIVE_19_CONTROL_POINT_PATCH = 26,
    D3D_PRIMITIVE_20_CONTROL_POINT_PATCH = 27,
    D3D_PRIMITIVE_21_CONTROL_POINT_PATCH = 28,
    D3D_PRIMITIVE_22_CONTROL_POINT_PATCH = 29,
    D3D_PRIMITIVE_23_CONTROL_POINT_PATCH = 30,
    D3D_PRIMITIVE_24_CONTROL_POINT_PATCH = 31,
```

```
D3D_PRIMITIVE_25_CONTROL_POINT_PATCH = 32,
D3D_PRIMITIVE_26_CONTROL_POINT_PATCH = 33,
D3D_PRIMITIVE_27_CONTROL_POINT_PATCH = 34,
D3D_PRIMITIVE_28_CONTROL_POINT_PATCH = 35,
D3D_PRIMITIVE_29_CONTROL_POINT_PATCH = 36,
D3D_PRIMITIVE_30_CONTROL_POINT_PATCH = 37,
D3D_PRIMITIVE_31_CONTROL_POINT_PATCH = 38,
D3D_PRIMITIVE_32_CONTROL_POINT_PATCH = 39,
D3D10_PRIMITIVE_UNDEFINED,
D3D10_PRIMITIVE_POINT,
D3D10_PRIMITIVE_LINE,
D3D10_PRIMITIVE_TRIANGLE,
D3D10_PRIMITIVE_LINE_ADJ,
D3D10_PRIMITIVE_TRIANGLE_ADJ,
D3D11_PRIMITIVE_UNDEFINED,
D3D11_PRIMITIVE_POINT,
D3D11_PRIMITIVE_LINE,
D3D11_PRIMITIVE_TRIANGLE,
D3D11_PRIMITIVE_LINE_ADJ,
D3D11_PRIMITIVE_TRIANGLE_ADJ,
D3D11_PRIMITIVE_1_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_2_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_3_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_4_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_5_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_6_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_7_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_8_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_9_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_10_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_11_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_12_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_13_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_14_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_15_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_16_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_17_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_18_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_19_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_20_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_21_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_22_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_23_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_24_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_25_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_26_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_27_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_28_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_29_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_30_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_31_CONTROL_POINT_PATCH,
D3D11_PRIMITIVE_32_CONTROL_POINT_PATCH
} ;
```

# Constants

[Expand table](#)

D3D_PRIMITIVE_UNDEFINED
Value: 0
D3D_PRIMITIVE_POINT
Value: 1
D3D_PRIMITIVE_LINE
Value: 2
D3D_PRIMITIVE_TRIANGLE
Value: 3
D3D_PRIMITIVE_LINE_ADJ
Value: 6
D3D_PRIMITIVE_TRIANGLE_ADJ
Value: 7
D3D_PRIMITIVE_1_CONTROL_POINT_PATCH
Value: 8
D3D_PRIMITIVE_2_CONTROL_POINT_PATCH
Value: 9
D3D_PRIMITIVE_3_CONTROL_POINT_PATCH
Value: 10
D3D_PRIMITIVE_4_CONTROL_POINT_PATCH
Value: 11
D3D_PRIMITIVE_5_CONTROL_POINT_PATCH
Value: 12
D3D_PRIMITIVE_6_CONTROL_POINT_PATCH
Value: 13
D3D_PRIMITIVE_7_CONTROL_POINT_PATCH
Value: 14
D3D_PRIMITIVE_8_CONTROL_POINT_PATCH
Value: 15
D3D_PRIMITIVE_9_CONTROL_POINT_PATCH
Value: 16

D3D\_PRIMITIVE\_10\_CONTROL\_POINT\_PATCH

Value: 17

D3D\_PRIMITIVE\_11\_CONTROL\_POINT\_PATCH

Value: 18

D3D\_PRIMITIVE\_12\_CONTROL\_POINT\_PATCH

Value: 19

D3D\_PRIMITIVE\_13\_CONTROL\_POINT\_PATCH

Value: 20

D3D\_PRIMITIVE\_14\_CONTROL\_POINT\_PATCH

Value: 21

D3D\_PRIMITIVE\_15\_CONTROL\_POINT\_PATCH

Value: 22

D3D\_PRIMITIVE\_16\_CONTROL\_POINT\_PATCH

Value: 23

D3D\_PRIMITIVE\_17\_CONTROL\_POINT\_PATCH

Value: 24

D3D\_PRIMITIVE\_18\_CONTROL\_POINT\_PATCH

Value: 25

D3D\_PRIMITIVE\_19\_CONTROL\_POINT\_PATCH

Value: 26

D3D\_PRIMITIVE\_20\_CONTROL\_POINT\_PATCH

Value: 27

D3D\_PRIMITIVE\_21\_CONTROL\_POINT\_PATCH

Value: 28

D3D\_PRIMITIVE\_22\_CONTROL\_POINT\_PATCH

Value: 29

D3D\_PRIMITIVE\_23\_CONTROL\_POINT\_PATCH

Value: 30

D3D\_PRIMITIVE\_24\_CONTROL\_POINT\_PATCH

Value: 31

D3D\_PRIMITIVE\_25\_CONTROL\_POINT\_PATCH

Value: 32

D3D\_PRIMITIVE\_26\_CONTROL\_POINT\_PATCH

Value: 33

D3D\_PRIMITIVE\_27\_CONTROL\_POINT\_PATCH

Value: 34

D3D\_PRIMITIVE\_28\_CONTROL\_POINT\_PATCH

Value: 35

D3D\_PRIMITIVE\_29\_CONTROL\_POINT\_PATCH

Value: 36

D3D\_PRIMITIVE\_30\_CONTROL\_POINT\_PATCH

Value: 37

D3D\_PRIMITIVE\_31\_CONTROL\_POINT\_PATCH

Value: 38

D3D\_PRIMITIVE\_32\_CONTROL\_POINT\_PATCH

Value: 39

D3D10\_PRIMITIVE\_UNDEFINED

D3D10\_PRIMITIVE\_POINT

D3D10\_PRIMITIVE\_LINE

D3D10\_PRIMITIVE\_TRIANGLE

D3D10\_PRIMITIVE\_LINE\_ADJ

D3D10\_PRIMITIVE\_TRIANGLE\_ADJ

D3D11\_PRIMITIVE\_UNDEFINED

The shader has not been initialized with an input primitive type.

D3D11\_PRIMITIVE\_POINT

Interpret the input primitive as a point.

D3D11\_PRIMITIVE\_LINE

Interpret the input primitive as a line.

D3D11\_PRIMITIVE\_TRIANGLE

Interpret the input primitive as a triangle.

D3D11\_PRIMITIVE\_LINE\_ADJ

Interpret the input primitive as a line with adjacency data.

D3D11\_PRIMITIVE\_TRIANGLE\_ADJ

Interpret the input primitive as a triangle with adjacency data.

**D3D11\_PRIMITIVE\_1\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_2\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_3\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_4\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_5\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_6\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_7\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_8\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_9\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_10\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_11\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_12\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_13\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_14\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_15\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_16\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

**D3D11\_PRIMITIVE\_17\_CONTROL\_POINT\_PATCH**

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_18\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_19\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_20\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_21\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_22\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_23\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_24\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_25\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_26\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_27\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_28\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_29\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_30\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_31\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

D3D11\_PRIMITIVE\_32\_CONTROL\_POINT\_PATCH

Interpret the input primitive as a control point patch.

## Remarks

The [ID3D11ShaderReflection::GetGSInputPrimitive](#) method returns a **D3D11\_PRIMITIVE**-typed value.

The **D3D11\_PRIMITIVE** enumeration is type defined in the D3D11.h header file as a [D3D\\_PRIMITIVE](#) enumeration, which is fully defined in the D3DCommon.h header file.

```
typedef D3D_PRIMITIVE D3D11_PRIMITIVE;
```

## Requirements

 [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Core Enumerations](#)

# D3D\_PRIMITIVE\_TOPOLOGY enumeration (d3dcommon.h)

Article 02/14/2023

Values that indicate how the pipeline interprets vertex data that is bound to the [input-assembler stage](#). These [primitive topology values](#) determine how the vertex data is rendered on screen.

## ⓘ Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_PRIMITIVE_TOPOLOGY {
    D3D_PRIMITIVE_TOPOLOGY_UNDEFINED = 0,
    D3D_PRIMITIVE_TOPOLOGY_POINTLIST = 1,
    D3D_PRIMITIVE_TOPOLOGY_LINELIST = 2,
    D3D_PRIMITIVE_TOPOLOGY_LINESTRIP = 3,
    D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST = 4,
    D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP = 5,
    D3D_PRIMITIVE_TOPOLOGY_TRIANGLEFAN,
    D3D_PRIMITIVE_TOPOLOGY_LINELIST_ADJ = 10,
    D3D_PRIMITIVE_TOPOLOGY_LINESTRIP_ADJ = 11,
    D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST_ADJ = 12,
    D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_ADJ = 13,
    D3D_PRIMITIVE_TOPOLOGY_1_CONTROL_POINT_PATCHLIST = 33,
    D3D_PRIMITIVE_TOPOLOGY_2_CONTROL_POINT_PATCHLIST = 34,
    D3D_PRIMITIVE_TOPOLOGY_3_CONTROL_POINT_PATCHLIST = 35,
    D3D_PRIMITIVE_TOPOLOGY_4_CONTROL_POINT_PATCHLIST = 36,
    D3D_PRIMITIVE_TOPOLOGY_5_CONTROL_POINT_PATCHLIST = 37,
    D3D_PRIMITIVE_TOPOLOGY_6_CONTROL_POINT_PATCHLIST = 38,
    D3D_PRIMITIVE_TOPOLOGY_7_CONTROL_POINT_PATCHLIST = 39,
    D3D_PRIMITIVE_TOPOLOGY_8_CONTROL_POINT_PATCHLIST = 40,
    D3D_PRIMITIVE_TOPOLOGY_9_CONTROL_POINT_PATCHLIST = 41,
    D3D_PRIMITIVE_TOPOLOGY_10_CONTROL_POINT_PATCHLIST = 42,
    D3D_PRIMITIVE_TOPOLOGY_11_CONTROL_POINT_PATCHLIST = 43,
    D3D_PRIMITIVE_TOPOLOGY_12_CONTROL_POINT_PATCHLIST = 44,
    D3D_PRIMITIVE_TOPOLOGY_13_CONTROL_POINT_PATCHLIST = 45,
    D3D_PRIMITIVE_TOPOLOGY_14_CONTROL_POINT_PATCHLIST = 46,
    D3D_PRIMITIVE_TOPOLOGY_15_CONTROL_POINT_PATCHLIST = 47,
```

```
D3D_PRIMITIVE_TOPOLOGY_16_CONTROL_POINT_PATCHLIST = 48,
D3D_PRIMITIVE_TOPOLOGY_17_CONTROL_POINT_PATCHLIST = 49,
D3D_PRIMITIVE_TOPOLOGY_18_CONTROL_POINT_PATCHLIST = 50,
D3D_PRIMITIVE_TOPOLOGY_19_CONTROL_POINT_PATCHLIST = 51,
D3D_PRIMITIVE_TOPOLOGY_20_CONTROL_POINT_PATCHLIST = 52,
D3D_PRIMITIVE_TOPOLOGY_21_CONTROL_POINT_PATCHLIST = 53,
D3D_PRIMITIVE_TOPOLOGY_22_CONTROL_POINT_PATCHLIST = 54,
D3D_PRIMITIVE_TOPOLOGY_23_CONTROL_POINT_PATCHLIST = 55,
D3D_PRIMITIVE_TOPOLOGY_24_CONTROL_POINT_PATCHLIST = 56,
D3D_PRIMITIVE_TOPOLOGY_25_CONTROL_POINT_PATCHLIST = 57,
D3D_PRIMITIVE_TOPOLOGY_26_CONTROL_POINT_PATCHLIST = 58,
D3D_PRIMITIVE_TOPOLOGY_27_CONTROL_POINT_PATCHLIST = 59,
D3D_PRIMITIVE_TOPOLOGY_28_CONTROL_POINT_PATCHLIST = 60,
D3D_PRIMITIVE_TOPOLOGY_29_CONTROL_POINT_PATCHLIST = 61,
D3D_PRIMITIVE_TOPOLOGY_30_CONTROL_POINT_PATCHLIST = 62,
D3D_PRIMITIVE_TOPOLOGY_31_CONTROL_POINT_PATCHLIST = 63,
D3D_PRIMITIVE_TOPOLOGY_32_CONTROL_POINT_PATCHLIST = 64,
D3D10_PRIMITIVE_TOPOLOGY_UNDEFINED,
D3D10_PRIMITIVE_TOPOLOGY_POINTLIST,
D3D10_PRIMITIVE_TOPOLOGY_LINELIST,
D3D10_PRIMITIVE_TOPOLOGY_LINESTRIP,
D3D10_PRIMITIVE_TOPOLOGY_TRIANGLELIST,
D3D10_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP,
D3D10_PRIMITIVE_TOPOLOGY_LINELIST_ADJ,
D3D10_PRIMITIVE_TOPOLOGY_LINESTRIP_ADJ,
D3D10_PRIMITIVE_TOPOLOGY_TRIANGLELIST_ADJ,
D3D10_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_ADJ,
D3D11_PRIMITIVE_TOPOLOGY_UNDEFINED,
D3D11_PRIMITIVE_TOPOLOGY_POINTLIST,
D3D11_PRIMITIVE_TOPOLOGY_LINELIST,
D3D11_PRIMITIVE_TOPOLOGY_LINESTRIP,
D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST,
D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP,
D3D11_PRIMITIVE_TOPOLOGY_LINELIST_ADJ,
D3D11_PRIMITIVE_TOPOLOGY_LINESTRIP_ADJ,
D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST_ADJ,
D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_ADJ,
D3D11_PRIMITIVE_TOPOLOGY_1_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_2_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_3_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_4_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_5_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_6_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_7_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_8_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_9_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_10_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_11_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_12_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_13_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_14_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_15_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_16_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_17_CONTROL_POINT_PATCHLIST,
D3D11_PRIMITIVE_TOPOLOGY_18_CONTROL_POINT_PATCHLIST,
```

```
D3D11_PRIMITIVE_TOPOLOGY_19_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_20_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_21_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_22_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_23_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_24_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_25_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_26_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_27_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_28_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_29_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_30_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_31_CONTROL_POINT_PATCHLIST,  
D3D11_PRIMITIVE_TOPOLOGY_32_CONTROL_POINT_PATCHLIST  
} ;
```

## Constants

`D3D_PRIMITIVE_TOPOLOGY_UNDEFINED`

Value: 0

The IA stage has not been initialized with a primitive topology. The IA stage will not function properly unless a primitive topology is defined.

`D3D_PRIMITIVE_TOPOLOGY_POINTLIST`

Value: 1

Interpret the vertex data as a list of points.

`D3D_PRIMITIVE_TOPOLOGY_LINELIST`

Value: 2

Interpret the vertex data as a list of lines.

`D3D_PRIMITIVE_TOPOLOGY_LINESTRIP`

Value: 3

Interpret the vertex data as a line strip.

`D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST`

Value: 4

Interpret the vertex data as a list of triangles.

`D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP`

Value: 5

Interpret the vertex data as a triangle strip.

`D3D_PRIMITIVE_TOPOLOGY_LINELIST_ADJ`

Value: 10

Interpret the vertex data as a list of lines with adjacency data.

`D3D_PRIMITIVE_TOPOLOGY_LINESTrip_Adj`

Value: 11

Interpret the vertex data as a line strip with adjacency data.

`D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST_Adj`

Value: 12

Interpret the vertex data as a list of triangles with adjacency data.

`D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_Adj`

Value: 13

Interpret the vertex data as a triangle strip with adjacency data.

`D3D_PRIMITIVE_TOPOLOGY_1_CONTROL_POINT_PATCHLIST`

Value: 33

Interpret the vertex data as a patch list.

`D3D_PRIMITIVE_TOPOLOGY_2_CONTROL_POINT_PATCHLIST`

Value: 34

Interpret the vertex data as a patch list.

`D3D_PRIMITIVE_TOPOLOGY_3_CONTROL_POINT_PATCHLIST`

Value: 35

Interpret the vertex data as a patch list.

`D3D_PRIMITIVE_TOPOLOGY_4_CONTROL_POINT_PATCHLIST`

Value: 36

Interpret the vertex data as a patch list.

`D3D_PRIMITIVE_TOPOLOGY_5_CONTROL_POINT_PATCHLIST`

Value: 37

Interpret the vertex data as a patch list.

`D3D_PRIMITIVE_TOPOLOGY_6_CONTROL_POINT_PATCHLIST`

Value: 38

Interpret the vertex data as a patch list.

`D3D_PRIMITIVE_TOPOLOGY_7_CONTROL_POINT_PATCHLIST`

Value: 39

Interpret the vertex data as a patch list.

`D3D_PRIMITIVE_TOPOLOGY_8_CONTROL_POINT_PATCHLIST`

Value: 40

Interpret the vertex data as a patch list.

`D3D_PRIMITIVE_TOPOLOGY_9_CONTROL_POINT_PATCHLIST`

Value: 41

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_10\_CONTROL\_POINT\_PATCHLIST

Value: 42

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_11\_CONTROL\_POINT\_PATCHLIST

Value: 43

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_12\_CONTROL\_POINT\_PATCHLIST

Value: 44

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_13\_CONTROL\_POINT\_PATCHLIST

Value: 45

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_14\_CONTROL\_POINT\_PATCHLIST

Value: 46

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_15\_CONTROL\_POINT\_PATCHLIST

Value: 47

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_16\_CONTROL\_POINT\_PATCHLIST

Value: 48

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_17\_CONTROL\_POINT\_PATCHLIST

Value: 49

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_18\_CONTROL\_POINT\_PATCHLIST

Value: 50

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_19\_CONTROL\_POINT\_PATCHLIST

Value: 51

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_20\_CONTROL\_POINT\_PATCHLIST

Value: 52

Interpret the vertex data as a patch list.

D3D\_PRIMITIVE\_TOPOLOGY\_21\_CONTROL\_POINT\_PATCHLIST

Value: 53

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_22\_CONTROL\_POINT\_PATCHLIST**

Value: 54

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_23\_CONTROL\_POINT\_PATCHLIST**

Value: 55

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_24\_CONTROL\_POINT\_PATCHLIST**

Value: 56

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_25\_CONTROL\_POINT\_PATCHLIST**

Value: 57

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_26\_CONTROL\_POINT\_PATCHLIST**

Value: 58

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_27\_CONTROL\_POINT\_PATCHLIST**

Value: 59

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_28\_CONTROL\_POINT\_PATCHLIST**

Value: 60

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_29\_CONTROL\_POINT\_PATCHLIST**

Value: 61

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_30\_CONTROL\_POINT\_PATCHLIST**

Value: 62

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_31\_CONTROL\_POINT\_PATCHLIST**

Value: 63

Interpret the vertex data as a patch list.

**D3D\_PRIMITIVE\_TOPOLOGY\_32\_CONTROL\_POINT\_PATCHLIST**

Value: 64

Interpret the vertex data as a patch list.

**D3D10\_PRIMITIVE\_TOPOLOGY\_UNDEFINED**

The IA stage has not been initialized with a primitive topology. The IA stage will not function properly unless a primitive topology is defined.

**D3D10\_PRIMITIVE\_TOPOLOGY\_POINTLIST**

Interpret the vertex data as a list of points.

**D3D10\_PRIMITIVE\_TOPOLOGY\_LINELIST**

Interpret the vertex data as a list of lines.

**D3D10\_PRIMITIVE\_TOPOLOGY\_LINESTRIP**

Interpret the vertex data as a line strip.

**D3D10\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST**

Interpret the vertex data as a list of triangles.

**D3D10\_PRIMITIVE\_TOPOLOGY\_TRIANGLESTRIP**

Interpret the vertex data as a triangle strip.

**D3D10\_PRIMITIVE\_TOPOLOGY\_LINELIST\_ADJ**

Interpret the vertex data as a list of lines with adjacency data.

**D3D10\_PRIMITIVE\_TOPOLOGY\_LINESTRIP\_ADJ**

Interpret the vertex data as a line strip with adjacency data.

**D3D10\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST\_ADJ**

Interpret the vertex data as a list of triangles with adjacency data.

**D3D10\_PRIMITIVE\_TOPOLOGY\_TRIANGLESTRIP\_ADJ**

Interpret the vertex data as a triangle strip with adjacency data.

**D3D11\_PRIMITIVE\_TOPOLOGY\_UNDEFINED**

The IA stage has not been initialized with a primitive topology. The IA stage will not function properly unless a primitive topology is defined.

**D3D11\_PRIMITIVE\_TOPOLOGY\_POINTLIST**

Interpret the vertex data as a list of points.

**D3D11\_PRIMITIVE\_TOPOLOGY\_LINELIST**

Interpret the vertex data as a list of lines.

**D3D11\_PRIMITIVE\_TOPOLOGY\_LINESTRIP**

Interpret the vertex data as a line strip.

**D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST**

Interpret the vertex data as a list of triangles.

**D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLESTRIP**

Interpret the vertex data as a triangle strip.

**D3D11\_PRIMITIVE\_TOPOLOGY\_LINELIST\_ADJ**

Interpret the vertex data as a list of lines with adjacency data.

**D3D11\_PRIMITIVE\_TOPOLOGY\_LINESTrip\_Adj**

Interpret the vertex data as a line strip with adjacency data.

**D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST\_Adj**

Interpret the vertex data as a list of triangles with adjacency data.

**D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLESTRIP\_Adj**

Interpret the vertex data as a triangle strip with adjacency data.

**D3D11\_PRIMITIVE\_TOPOLOGY\_1\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_2\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_3\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_4\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_5\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_6\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_7\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_8\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_9\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_10\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_11\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_12\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_13\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_14\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_15\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_16\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_17\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_18\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_19\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_20\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_21\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_22\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_23\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_24\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_25\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_26\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_27\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_28\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

**D3D11\_PRIMITIVE\_TOPOLOGY\_29\_CONTROL\_POINT\_PATCHLIST**

Interpret the vertex data as a patch list.

`D3D11_PRIMITIVE_TOPOLOGY_30_CONTROL_POINT_PATCHLIST`

Interpret the vertex data as a patch list.

`D3D11_PRIMITIVE_TOPOLOGY_31_CONTROL_POINT_PATCHLIST`

Interpret the vertex data as a patch list.

`D3D11_PRIMITIVE_TOPOLOGY_32_CONTROL_POINT_PATCHLIST`

Interpret the vertex data as a patch list.

## Remarks

Use the [ID3D11DeviceContext::IASetPrimitiveTopology](#) method and a value from `D3D_PRIMITIVE_TOPOLOGY` to bind a primitive topology to the input-assembler stage. Use the [ID3D11DeviceContext::IAGetPrimitiveTopology](#) method to retrieve the primitive topology for the input-assembler stage.

The following diagram shows the various primitive types for a geometry shader object.

# Geometry Shader Input Primitives

Legend:

- = Vertex



$v[n][e]$  = Input vertex  $n$ , element  $e$  ( $n$  and  $e$  independently indexable)

POINT

$v[0][e]$

LINE

$v[0][e]$  —  $v[1][e]$

LINE\_ADJ

$v[0][e]$  —  $v[1][e]$  —  $v[2][e]$  —  $v[3][e]$

TRIANGLE

A triangle with vertices labeled  $v[0][e]$ ,  $v[1][e]$ , and  $v[2][e]$ . The top vertex has a winding direction arrow pointing clockwise.

TRIANGLE\_ADJ

A triangle with vertices labeled  $v[0][e]$ ,  $v[1][e]$ ,  $v[2][e]$ ,  $v[3][e]$ ,  $v[4][e]$ , and  $v[5][e]$ . The top vertex has a winding direction arrow pointing clockwise. Dashed lines connect  $v[0][e]$  to  $v[1][e]$  and  $v[2][e]$ , and  $v[1][e]$  to  $v[3][e]$ .

1-32 CONTROL POINT PATCH

$v[0][e]$

•

$v[n][e]$

Additional Input

vPrim: 32bit scalar, available for all input primitives. Generated by Input Assembler as "PrimitiveID". Only single value, for the interior primitive, not any adjacent primitives.

## Requirements

Header

d3dcommon.h

## See also

[Common Version Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D\_REGISTER\_COMPONENT\_TYPE enumeration (d3dcommon.h)

Article 02/22/2024

Values that identify the data types that can be stored in a register.

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_REGISTER_COMPONENT_TYPE {
    D3D_REGISTER_COMPONENT_UNKNOWN = 0,
    D3D_REGISTER_COMPONENT_UINT32 = 1,
    D3D_REGISTER_COMPONENT_SINT32 = 2,
    D3D_REGISTER_COMPONENT_FLOAT32 = 3,
    D3D_REGISTER_COMPONENT_UINT16,
    D3D_REGISTER_COMPONENT_SINT16,
    D3D_REGISTER_COMPONENT_FLOAT16,
    D3D_REGISTER_COMPONENT_UINT64,
    D3D_REGISTER_COMPONENT_SINT64,
    D3D_REGISTER_COMPONENT_FLOAT64,
    D3D10_REGISTER_COMPONENT_UNKNOWN,
    D3D10_REGISTER_COMPONENT_UINT32,
    D3D10_REGISTER_COMPONENT_SINT32,
    D3D10_REGISTER_COMPONENT_FLOAT32,
    D3D10_REGISTER_COMPONENT_UINT16,
    D3D10_REGISTER_COMPONENT_SINT16,
    D3D10_REGISTER_COMPONENT_FLOAT16,
    D3D10_REGISTER_COMPONENT_UINT64,
    D3D10_REGISTER_COMPONENT_SINT64,
    D3D10_REGISTER_COMPONENT_FLOAT64
} ;
```

## Constants

 Expand table

`D3D_REGISTER_COMPONENT_UNKNOWN`

Value: 0

The data type is unknown.

`D3D_REGISTER_COMPONENT_UINT32`

Value: 1

32-bit unsigned integer.

`D3D_REGISTER_COMPONENT_SINT32`

Value: 2

32-bit signed integer.

`D3D_REGISTER_COMPONENT_FLOAT32`

Value: 3

32-bit floating-point number.

`D3D10_REGISTER_COMPONENT_UNKNOWN`

The data type is unknown.

`D3D10_REGISTER_COMPONENT_UINT32`

32-bit unsigned integer.

`D3D10_REGISTER_COMPONENT_SINT32`

32-bit signed integer.

`D3D10_REGISTER_COMPONENT_FLOAT32`

32-bit floating-point number.

## Remarks

A register component type is specified in the `ComponentType` member of the [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#) structure.

## Requirements

 Expand table

Requirement	Value
Header	d3dcommon.h

## See also

## Common Version Enumerations

# D3D\_RESOURCE\_RETURN\_TYPE enumeration (d3dcommon.h)

Article01/31/2022

Indicates return value type.

## ⓘ Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_RESOURCE_RETURN_TYPE {
    D3D_RETURN_TYPE_UNORM = 1,
    D3D_RETURN_TYPE_SNORM = 2,
    D3D_RETURN_TYPE_SINT = 3,
    D3D_RETURN_TYPE_UINT = 4,
    D3D_RETURN_TYPE_FLOAT = 5,
    D3D_RETURN_TYPE_MIXED = 6,
    D3D_RETURN_TYPE_DOUBLE = 7,
    D3D_RETURN_TYPE_CONTINUED = 8,
    D3D10_RETURN_TYPE_UNORM,
    D3D10_RETURN_TYPE_SNORM,
    D3D10_RETURN_TYPE_SINT,
    D3D10_RETURN_TYPE_UINT,
    D3D10_RETURN_TYPE_FLOAT,
    D3D10_RETURN_TYPE_MIXED,
    D3D11_RETURN_TYPE_UNORM,
    D3D11_RETURN_TYPE_SNORM,
    D3D11_RETURN_TYPE_SINT,
    D3D11_RETURN_TYPE_UINT,
    D3D11_RETURN_TYPE_FLOAT,
    D3D11_RETURN_TYPE_MIXED,
    D3D11_RETURN_TYPE_DOUBLE,
    D3D11_RETURN_TYPE_CONTINUED
} ;
```

## Constants

[Expand table](#)

D3D\_RETURN\_TYPE\_UNORM

Value: 1

D3D\_RETURN\_TYPE\_SNORM

Value: 2

D3D\_RETURN\_TYPE\_SINT

Value: 3

D3D\_RETURN\_TYPE\_UINT

Value: 4

D3D\_RETURN\_TYPE\_FLOAT

Value: 5

D3D\_RETURN\_TYPE\_MIXED

Value: 6

D3D\_RETURN\_TYPE\_DOUBLE

Value: 7

D3D10\_RETURN\_TYPE\_CONTINUED

Value: 8

D3D10\_RETURN\_TYPE\_UNORM

D3D10\_RETURN\_TYPE\_SNORM

D3D10\_RETURN\_TYPE\_SINT

D3D10\_RETURN\_TYPE\_UINT

D3D10\_RETURN\_TYPE\_FLOAT

D3D10\_RETURN\_TYPE\_MIXED

D3D11\_RETURN\_TYPE\_UNORM

Return type is UNORM.

D3D11\_RETURN\_TYPE\_SNORM

Return type is SNORM.

D3D11\_RETURN\_TYPE\_SINT

Return type is SINT.

D3D11\_RETURN\_TYPE\_UINT

Return type is UINT.

**D3D11\_RETURN\_TYPE\_FLOAT**

Return type is FLOAT.

**D3D11\_RETURN\_TYPE\_MIXED**

Return type is unknown.

**D3D11\_RETURN\_TYPE\_DOUBLE**

Return type is DOUBLE.

**D3D11\_RETURN\_TYPE\_CONTINUED**

Return type is a multiple-dword type, such as a double or uint64, and the component is continued from the previous component that was declared. The first component represents the lower bits.

## Remarks

The `D3D11_RESOURCE_RETURN_TYPE` enumeration is type defined in the `D3D11shader.h` header file as a `D3D_RESOURCE_RETURN_TYPE` enumeration, which is fully defined in the `D3DCCommon.h` header file.

```
typedef D3D_RESOURCE_RETURN_TYPE D3D11_RESOURCE_RETURN_TYPE;
```

## Requirements

 Expand table

Requirement	Value
Header	d3dcommon.h

## See also

[Shader Enumerations](#)

# D3D\_SHADER\_CBUFFER\_FLAGS enumeration (d3dcommon.h)

Article 02/22/2024

Values that identify the intended use of a constant-data buffer.

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum _D3D_SHADER_CBUFFER_FLAGS {
    D3D_CBF_USERPACKED = 1,
    D3D10_CBF_USERPACKED,
    D3D_CBF_FORCE_DWORD = 0x7fffffff
} D3D_SHADER_CBUFFER_FLAGS;
```

## Constants

 Expand table

`D3D_CBF_USERPACKED`

Value: 1

Bind the constant buffer to an input slot defined in HLSL code (instead of letting the compiler choose the input slot).

`D3D10_CBF_USERPACKED`

Bind the constant buffer to an input slot defined in HLSL code (instead of letting the compiler choose the input slot).

`D3D_CBF_FORCE_DWORD`

Value: `0x7fffffff`

This value is not used by a programmer; it exists to force the enumeration to compile to 32 bits.

## Remarks

D3D\_SHADER\_CBUFFER\_FLAGS-typed values are specified in the [uFlags](#) member of the [D3D11\\_SHADER\\_BUFFER\\_DESC](#) structure.

## Requirements

 [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

# D3D\_SHADER\_INPUT\_FLAGS enumeration (d3dcommon.h)

Article 02/22/2024

Values that identify shader-input options.

## ⚠ Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum _D3D_SHADER_INPUT_FLAGS {
    D3D_SIF_USERPACKED = 0x1,
    D3D_SIF_COMPARISON_SAMPLER = 0x2,
    D3D_SIF_TEXTURE_COMPONENT_0 = 0x4,
    D3D_SIF_TEXTURE_COMPONENT_1 = 0x8,
    D3D_SIF_TEXTURE_COMPONENTS = 0xc,
    D3D_SIF_UNUSED = 0x10,
    D3D10_SIF_USERPACKED,
    D3D10_SIF_COMPARISON_SAMPLER,
    D3D10_SIF_TEXTURE_COMPONENT_0,
    D3D10_SIF_TEXTURE_COMPONENT_1,
    D3D10_SIF_TEXTURE_COMPONENTS,
    D3D_SIF_FORCE_DWORD = 0x7fffffff
} D3D_SHADER_INPUT_FLAGS;
```

## Constants

[ ] Expand table

<code>D3D_SIF_USERPACKED</code>
Value: <code>0x1</code>
Assign a shader input to a register based on the register assignment in the HLSL code (instead of letting the compiler choose the register).

**D3D\_SIF\_COMPARISON\_SAMPLER**

Value: *0x2*

Use a comparison sampler, which uses the [SampleCmp \(DirectX HLSL Texture Object\)](#) and [SampleCmpLevelZero \(DirectX HLSL Texture Object\)](#) sampling functions.

**D3D\_SIF\_TEXTURE\_COMPONENT\_0**

Value: *0x4*

A 2-bit value for encoding texture components.

**D3D\_SIF\_TEXTURE\_COMPONENT\_1**

Value: *0x8*

A 2-bit value for encoding texture components.

**D3D\_SIF\_TEXTURE\_COMPONENTS**

Value: *0xc*

A 2-bit value for encoding texture components.

**D3D\_SIF\_UNUSED**

Value: *0x10*

This value is reserved.

**D3D10\_SIF\_USERPACKED**

Assign a shader input to a register based on the register assignment in the HLSL code (instead of letting the compiler choose the register).

**D3D10\_SIF\_COMPARISON\_SAMPLER**

Use a comparison sampler, which uses the [SampleCmp \(DirectX HLSL Texture Object\)](#) and [SampleCmpLevelZero \(DirectX HLSL Texture Object\)](#) sampling functions.

**D3D10\_SIF\_TEXTURE\_COMPONENT\_0**

A 2-bit value for encoding texture components.

**D3D10\_SIF\_TEXTURE\_COMPONENT\_1**

A 2-bit value for encoding texture components.

**D3D10\_SIF\_TEXTURE\_COMPONENTS**

A 2-bit value for encoding texture components.

**D3D\_SIF\_FORCE\_DWORD**

Value: *0xffffffff*

Forces the enumeration to compile to 32 bits.

This value is not used directly by titles.

## Remarks

**D3D\_SHADER\_INPUT\_FLAGS**-typed values are specified in the **uFlags** member of the [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC](#) structure.

# Requirements

 [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

# D3D\_SHADER\_INPUT\_TYPE enumeration (d3dcommon.h)

Article01/31/2022

Values that identify resource types that can be bound to a shader and that are reflected as part of the resource description for the shader.

## ⓘ Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum _D3D_SHADER_INPUT_TYPE {
    D3D_SIT_CBUFFER = 0,
    D3D_SIT_TBUFFER,
    D3D_SIT_TEXTURE,
    D3D_SIT_SAMPLER,
    D3D_SIT_UAV_RWTYPED,
    D3D_SIT_STRUCTURED,
    D3D_SIT_UAV_RWSTRUCTURED,
    D3D_SIT_BYTEADDRESS,
    D3D_SIT_UAV_RWBYTEADDRESS,
    D3D_SIT_UAV_APPEND_STRUCTURED,
    D3D_SIT_UAV_CONSUME_STRUCTURED,
    D3D_SIT_UAV_RWSTRUCTURED_WITH_COUNTER,
    D3D_SIT_RTACCELERATIONSTRUCTURE,
    D3D_SIT_UAV_FEEDBACKTEXTURE,
    D3D10_SIT_CBUFFER,
    D3D10_SIT_TBUFFER,
    D3D10_SIT_TEXTURE,
    D3D10_SIT_SAMPLER,
    D3D11_SIT_UAV_RWTYPED,
    D3D11_SIT_STRUCTURED,
    D3D11_SIT_UAV_RWSTRUCTURED,
    D3D11_SIT_BYTEADDRESS,
    D3D11_SIT_UAV_RWBYTEADDRESS,
    D3D11_SIT_UAV_APPEND_STRUCTURED,
    D3D11_SIT_UAV_CONSUME_STRUCTURED,
    D3D11_SIT_UAV_RWSTRUCTURED_WITH_COUNTER
} D3D_SHADER_INPUT_TYPE;
```

# Constants

[Expand table](#)

<b>D3D_SIT_CBUFFER</b> Value: 0 The shader resource is a constant buffer.
<b>D3D_SIT_TBUFFER</b> The shader resource is a texture buffer.
<b>D3D_SIT_TEXTURE</b> The shader resource is a texture.
<b>D3D_SIT_SAMPLER</b> The shader resource is a sampler.
<b>D3D_SIT_UAV_RW_TYPED</b> The shader resource is a read-and-write buffer or texture.
<b>D3D_SIT_STRUCTURED</b> The shader resource is a structured buffer.  For more information about structured buffer, see the <b>Remarks</b> section.
<b>D3D_SIT_UAV_RWSTRUCTURED</b> The shader resource is a read-and-write structured buffer.
<b>D3D_SIT_BYTEADDRESS</b> The shader resource is a byte-address buffer.
<b>D3D_SIT_UAV_RWBYTEADDRESS</b> The shader resource is a read-and-write byte-address buffer.
<b>D3D_SIT_UAV_APPEND_STRUCTURED</b> The shader resource is an append-structured buffer.
<b>D3D_SIT_UAV_CONSUME_STRUCTURED</b> The shader resource is a consume-structured buffer.
<b>D3D_SIT_UAV_RWSTRUCTURED_WITH_COUNTER</b> The shader resource is a read-and-write structured buffer that uses the built-in counter to append or consume.
<b>D3D10_SIT_CBUFFER</b> The shader resource is a constant buffer.

**D3D10\_SIT\_TBUFFER**

The shader resource is a texture buffer.

**D3D10\_SIT\_TEXTURE**

The shader resource is a texture.

**D3D10\_SIT\_SAMPLER**

The shader resource is a sampler.

**D3D11\_SIT\_UAV\_RW\_TYPED**

The shader resource is a read-and-write buffer.

**D3D11\_SIT\_STRUCTURED**

The shader resource is a structured buffer.

For more information about structured buffer, see the **Remarks** section.

**D3D11\_SIT\_UAV\_RWSTRUCTURED**

The shader resource is a read-and-write structured buffer.

**D3D11\_SIT\_BYTEADDRESS**

The shader resource is a byte-address buffer.

**D3D11\_SIT\_UAV\_RWBYTEADDRESS**

The shader resource is a read-and-write byte-address buffer.

**D3D11\_SIT\_UAV\_APPEND\_STRUCTURED**

The shader resource is an append-structured buffer.

**D3D11\_SIT\_UAV\_CONSUME\_STRUCTURED**

The shader resource is a consume-structured buffer.

**D3D11\_SIT\_UAV\_RWSTRUCTURED\_WITH\_COUNTER**

The shader resource is a read-and-write structured buffer that uses the built-in counter to append or consume.

## Remarks

`D3D_SHADER_INPUT_TYPE`-typed values are specified in the **Type** member of the [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC](#) structure.

## Requirements

 Expand table

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

# D3D\_SHADER\_VARIABLE\_CLASS enumeration (d3dcommon.h)

Article01/31/2022

Values that identify the class of a shader variable.

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum _D3D_SHADER_VARIABLE_CLASS {
    D3D_SVC_SCALAR = 0,
    D3D_SVC_VECTOR,
    D3D_SVC_MATRIX_ROWS,
    D3D_SVC_MATRIX_COLUMNS,
    D3D_SVC_OBJECT,
    D3D_SVC_STRUCT,
    D3D_SVC_INTERFACE_CLASS,
    D3D_SVC_INTERFACE_POINTER,
    D3D10_SVC_SCALAR,
    D3D10_SVC_VECTOR,
    D3D10_SVC_MATRIX_ROWS,
    D3D10_SVC_MATRIX_COLUMNS,
    D3D10_SVC_OBJECT,
    D3D10_SVC_STRUCT,
    D3D11_SVC_INTERFACE_CLASS,
    D3D11_SVC_INTERFACE_POINTER,
    D3D_SVC_FORCE_DWORD = 0x7fffffff
} D3D_SHADER_VARIABLE_CLASS;
```

## Constants

[+] Expand table

**D3D\_SVC\_SCALAR**

Value: 0

The shader variable is a scalar.

**D3D\_SVC\_VECTOR**

The shader variable is a vector.

**D3D\_SVC\_MATRIX\_ROWS**

The shader variable is a row-major matrix.

**D3D\_SVC\_MATRIX\_COLUMNS**

The shader variable is a column-major matrix.

**D3D\_SVC\_OBJECT**

The shader variable is an object.

**D3D\_SVC\_STRUCT**

The shader variable is a structure.

**D3D\_SVC\_INTERFACE\_CLASS**

The shader variable is a class.

**D3D\_SVC\_INTERFACE\_POINTER**

The shader variable is an interface.

**D3D10\_SVC\_SCALAR**

The shader variable is a scalar.

**D3D10\_SVC\_VECTOR**

The shader variable is a vector.

**D3D10\_SVC\_MATRIX\_ROWS**

The shader variable is a row-major matrix.

**D3D10\_SVC\_MATRIX\_COLUMNS**

The shader variable is a column-major matrix.

**D3D10\_SVC\_OBJECT**

The shader variable is an object.

**D3D10\_SVC\_STRUCT**

The shader variable is a structure.

**D3D11\_SVC\_INTERFACE\_CLASS**

The shader variable is a class.

**D3D11\_SVC\_INTERFACE\_POINTER**

The shader variable is an interface.

`D3D_SVC_FORCE_DWORD`

Value: `0x7fffffff`

This value is not used by a programmer; it exists to force the enumeration to compile to 32 bits.

## Remarks

The class of a shader variable is not a programming class; the class identifies the variable class such as scalar, vector, object, and so on. `D3D_SHADER_VARIABLE_CLASS`-typed values are specified in the `Class` member of the `D3D11_SHADER_TYPE_DESC` structure.

## Requirements

 Expand table

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

# D3D\_SHADER\_VARIABLE\_FLAGS enumeration (d3dcommon.h)

Article 02/22/2024

Values that identify information about a shader variable.

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum _D3D_SHADER_VARIABLE_FLAGS {
    D3D_SVF_USERPACKED = 1,
    D3D_SVF_USED = 2,
    D3D_SVF_INTERFACE_POINTER = 4,
    D3D_SVF_INTERFACE_PARAMETER = 8,
    D3D10_SVF_USERPACKED,
    D3D10_SVF_USED,
    D3D11_SVF_INTERFACE_POINTER,
    D3D11_SVF_INTERFACE_PARAMETER,
    D3D_SVF_FORCE_DWORD = 0x7fffffff
} D3D_SHADER_VARIABLE_FLAGS;
```

## Constants

 Expand table

`D3D_SVF_USERPACKED`

Value: 1

Indicates that the registers assigned to this shader variable were explicitly declared in shader code (instead of automatically assigned by the compiler).

`D3D_SVF_USED`

Value: 2

Indicates that this variable is used by this shader. This value confirms that a particular shader variable (which can be common to many different shaders) is indeed used by a particular shader.

**D3D\_SVF\_INTERFACE\_POINTER**

Value: 4

Indicates that this variable is an interface.

**D3D\_SVF\_INTERFACE\_PARAMETER**

Value: 8

Indicates that this variable is a parameter of an interface.

**D3D10\_SVF\_USERPACKED**

Indicates that the registers assigned to this shader variable were explicitly declared in shader code (instead of automatically assigned by the compiler).

**D3D10\_SVF\_USED**

Indicates that this variable is used by this shader. This value confirms that a particular shader variable (which can be common to many different shaders) is indeed used by a particular shader.

**D3D11\_SVF\_INTERFACE\_POINTER**

Indicates that this variable is an interface.

**D3D11\_SVF\_INTERFACE\_PARAMETER**

Indicates that this variable is a parameter of an interface.

**D3D\_SVF\_FORCE\_DWORD**

Value: 0x7fffffff

This value is not used by a programmer; it exists to force the enumeration to compile to 32 bits.

## Remarks

A call to the [ID3D11ShaderReflectionVariable::GetDesc](#) method returns **D3D\_SHADER\_VARIABLE\_FLAGS** values in the **uFlags** member of a [D3D11\\_SHADER\\_VARIABLE\\_DESC](#) structure.

## Requirements

 Expand table

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

# D3D\_SHADER\_VARIABLE\_TYPE enumeration (d3dcommon.h)

Article01/31/2022

Values that identify various data, texture, and buffer types that can be assigned to a shader variable.

## ! Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum _D3D_SHADER_VARIABLE_TYPE {
    D3D_SVT_VOID = 0,
    D3D_SVT_BOOL = 1,
    D3D_SVT_INT = 2,
    D3D_SVT_FLOAT = 3,
    D3D_SVT_STRING = 4,
    D3D_SVT_TEXTURE = 5,
    D3D_SVT_TEXTURE1D = 6,
    D3D_SVT_TEXTURE2D = 7,
    D3D_SVT_TEXTURE3D = 8,
    D3D_SVT_TEXTURECUBE = 9,
    D3D_SVT_SAMPLER = 10,
    D3D_SVT_SAMPLER1D = 11,
    D3D_SVT_SAMPLER2D = 12,
    D3D_SVT_SAMPLER3D = 13,
    D3D_SVT_SAMPLERCUBE = 14,
    D3D_SVT_PIXELSHADER = 15,
    D3D_SVT_VERTEXSHADER = 16,
    D3D_SVT_PIXELFRAGMENT = 17,
    D3D_SVT_VERTEXFRAGMENT = 18,
    D3D_SVT_UINT = 19,
    D3D_SVT_UINT8 = 20,
    D3D_SVT_GEOMETRYSHADER = 21,
    D3D_SVT_RASTERIZER = 22,
    D3D_SVT_DEPTHSTENCIL = 23,
    D3D_SVT_BLEND = 24,
    D3D_SVT_BUFFER = 25,
    D3D_SVT_CBUFFER = 26,
    D3D_SVT_TBUFFER = 27,
```

```
D3D_SVT_TEXTURE1DARRAY = 28,
D3D_SVT_TEXTURE2DARRAY = 29,
D3D_SVT_RENDERTARGETVIEW = 30,
D3D_SVT_DEPTHSTENCILVIEW = 31,
D3D_SVT_TEXTURE2DMS = 32,
D3D_SVT_TEXTURE2DMSARRAY = 33,
D3D_SVT_TEXTURECUBEARRAY = 34,
D3D_SVT_HULLSHADER = 35,
D3D_SVT_DOMAINSHADER = 36,
D3D_SVT_INTERFACE_POINTER = 37,
D3D_SVT_COMPUTESHADER = 38,
D3D_SVT_DOUBLE = 39,
D3D_SVT_RWTEXTURE1D = 40,
D3D_SVT_RWTEXTURE1DARRAY = 41,
D3D_SVT_RWTEXTURE2D = 42,
D3D_SVT_RWTEXTURE2DARRAY = 43,
D3D_SVT_RWTEXTURE3D = 44,
D3D_SVT_RWBUFFER = 45,
D3D_SVT_BYTEADDRESS_BUFFER = 46,
D3D_SVT_RWBYTEADDRESS_BUFFER = 47,
D3D_SVT_STRUCTURED_BUFFER = 48,
D3D_SVT_RWSTRUCTURED_BUFFER = 49,
D3D_SVT_APPEND_STRUCTURED_BUFFER = 50,
D3D_SVT_CONSUME_STRUCTURED_BUFFER = 51,
D3D_SVT_MIN8FLOAT = 52,
D3D_SVT_MIN10FLOAT = 53,
D3D_SVT_MIN16FLOAT = 54,
D3D_SVT_MIN12INT = 55,
D3D_SVT_MIN16INT = 56,
D3D_SVT_MIN16UINT = 57,
D3D_SVT_INT16,
D3D_SVT_UINT16,
D3D_SVT_FLOAT16,
D3D_SVT_INT64,
D3D_SVT_UINT64,
D3D10_SVT_VOID,
D3D10_SVT_BOOL,
D3D10_SVT_INT,
D3D10_SVT_FLOAT,
D3D10_SVT_STRING,
D3D10_SVT_TEXTURE,
D3D10_SVT_TEXTURE1D,
D3D10_SVT_TEXTURE2D,
D3D10_SVT_TEXTURE3D,
D3D10_SVT_TEXTURECUBE,
D3D10_SVT_SAMPLER,
D3D10_SVT_SAMPLER1D,
D3D10_SVT_SAMPLER2D,
D3D10_SVT_SAMPLER3D,
D3D10_SVT_SAMPLERCUBE,
D3D10_SVT_PIXELSHADER,
D3D10_SVT_VERTEXSHADER,
D3D10_SVT_PIXELFRAGMENT,
D3D10_SVT_VERTEXFRAGMENT,
D3D10_SVT_UINT,
```

```
D3D10_SVT_UINT8,
D3D10_SVT_GEOMETRYSHADER,
D3D10_SVT_RASTERIZER,
D3D10_SVT_DEPTHSTENCIL,
D3D10_SVT_BLEND,
D3D10_SVT_BUFFER,
D3D10_SVT_CBUFFER,
D3D10_SVT_TBUFFER,
D3D10_SVT_TEXTURE1DARRAY,
D3D10_SVT_TEXTURE2DARRAY,
D3D10_SVT_RENDERTARGETVIEW,
D3D10_SVT_DEPTHSTENCILVIEW,
D3D10_SVT_TEXTURE2DMS,
D3D10_SVT_TEXTURE2DMSARRAY,
D3D10_SVT_TEXTURECUBEARRAY,
D3D11_SVT_HULLSHADER,
D3D11_SVT_DOMAINSHADER,
D3D11_SVT_INTERFACE_POINTER,
D3D11_SVT_COMPUTESHADER,
D3D11_SVT_DOUBLE,
D3D11_SVT_RWTEXTURE1D,
D3D11_SVT_RWTEXTURE1DARRAY,
D3D11_SVT_RWTEXTURE2D,
D3D11_SVT_RWTEXTURE2DARRAY,
D3D11_SVT_RWTEXTURE3D,
D3D11_SVT_RWBUFFER,
D3D11_SVT_BYTEADDRESS_BUFFER,
D3D11_SVT_RWBYTEADDRESS_BUFFER,
D3D11_SVT_STRUCTURED_BUFFER,
D3D11_SVT_RWSTRUCTURED_BUFFER,
D3D11_SVT_APPEND_STRUCTURED_BUFFER,
D3D11_SVT_CONSUME_STRUCTURED_BUFFER,
D3D_SVT_FORCE_DWORD = 0x7fffffff
} D3D_SHADER_VARIABLE_TYPE;
```

## Constants

[+] Expand table

D3D\_SVT\_VOID

Value: 0

The variable is a void pointer.

D3D\_SVT\_BOOL

Value: 1

The variable is a boolean.

D3D\_SVT\_INT

Value: 2

The variable is an integer.

D3D\_SVT\_FLOAT

Value: 3

The variable is a floating-point number.

D3D\_SVT\_STRING

Value: 4

The variable is a string.

D3D\_SVT\_TEXTURE

Value: 5

The variable is a texture.

D3D\_SVT\_TEXTURE1D

Value: 6

The variable is a 1D texture.

D3D\_SVT\_TEXTURE2D

Value: 7

The variable is a 2D texture.

D3D\_SVT\_TEXTURE3D

Value: 8

The variable is a 3D texture.

D3D\_SVT\_TEXTURECUBE

Value: 9

The variable is a texture cube.

D3D\_SVT\_SAMPLER

Value: 10

The variable is a sampler.

D3D\_SVT\_SAMPLER1D

Value: 11

The variable is a 1D sampler.

D3D\_SVT\_SAMPLER2D

Value: 12

The variable is a 2D sampler.

D3D\_SVT\_SAMPLER3D

Value: 13

The variable is a 3D sampler.

D3D\_SVT\_SAMPLERCUBE

Value: 14

The variable is a cube sampler.

**D3D\_SVT\_PIXELSHADER**

Value: 15

The variable is a pixel shader.

**D3D\_SVT\_VERTEXSHADER**

Value: 16

The variable is a vertex shader.

**D3D\_SVT\_PIXELFRAGMENT**

Value: 17

The variable is a pixel fragment.

**D3D\_SVT\_VERTEXFRAGMENT**

Value: 18

The variable is a vertex fragment.

**D3D\_SVT\_UINT**

Value: 19

The variable is an unsigned integer.

**D3D\_SVT\_UINT8**

Value: 20

The variable is an 8-bit unsigned integer.

**D3D\_SVT\_GEOMETRYSHADER**

Value: 21

The variable is a geometry shader.

**D3D\_SVT\_RASTERIZER**

Value: 22

The variable is a rasterizer-state object.

**D3D\_SVT\_DEPTHSTENCIL**

Value: 23

The variable is a depth-stencil-state object.

**D3D\_SVT\_BLEND**

Value: 24

The variable is a blend-state object.

**D3D\_SVT\_BUFFER**

Value: 25

The variable is a buffer.

**D3D\_SVT\_CBUFFER**

Value: 26

The variable is a constant buffer.

`D3D_SVT_TBUFFER`

Value: 27

The variable is a texture buffer.

`D3D_SVT_TEXTURE1DARRAY`

Value: 28

The variable is a 1D-texture array.

`D3D_SVT_TEXTURE2DARRAY`

Value: 29

The variable is a 2D-texture array.

`D3D_SVT_RENDERTARGETVIEW`

Value: 30

The variable is a render-target view.

`D3D_SVT_DEPTHSTENCILVIEW`

Value: 31

The variable is a depth-stencil view.

`D3D_SVT_TEXTURE2DMS`

Value: 32

The variable is a 2D-multisampled texture.

`D3D_SVT_TEXTURE2DMSARRAY`

Value: 33

The variable is a 2D-multisampled-texture array.

`D3D_SVT_TEXTURECUBEARRAY`

Value: 34

The variable is a texture-cube array.

`D3D_SVT_HULLSHADER`

Value: 35

The variable holds a compiled hull-shader binary.

`D3D_SVT_DOMAINSHADER`

Value: 36

The variable holds a compiled domain-shader binary.

`D3D_SVT_INTERFACE_POINTER`

Value: 37

The variable is an interface.

`D3D_SVT_COMPUTESHADER`

Value: 38

The variable holds a compiled compute-shader binary.

`D3D_SVT_DOUBLE`

Value: 39

The variable is a double precision (64-bit) floating-point number.

`D3D_SVT_RWTEXTURE1D`

Value: 40

The variable is a 1D read-and-write texture.

`D3D_SVT_RWTEXTURE1DARRAY`

Value: 41

The variable is an array of 1D read-and-write textures.

`D3D_SVT_RWTEXTURE2D`

Value: 42

The variable is a 2D read-and-write texture.

`D3D_SVT_RWTEXTURE2DARRAY`

Value: 43

The variable is an array of 2D read-and-write textures.

`D3D_SVT_RWTEXTURE3D`

Value: 44

The variable is a 3D read-and-write texture.

`D3D_SVT_RWBUFFER`

Value: 45

The variable is a read-and-write buffer.

`D3D_SVT_BYTEADDRESS_BUFFER`

Value: 46

The variable is a byte-address buffer.

`D3D_SVT_RWBYTEADDRESS_BUFFER`

Value: 47

The variable is a read-and-write byte-address buffer.

`D3D_SVT_STRUCTURED_BUFFER`

Value: 48

The variable is a structured buffer.

For more information about structured buffer, see the **Remarks** section.

`D3D_SVT_RWSTRUCTURED_BUFFER`

Value: 49

The variable is a read-and-write structured buffer.

`D3D_SVT_APPEND_STRUCTURED_BUFFER`

Value: 50

The variable is an append structured buffer.

D3D\_SVT\_CONSUME\_STRUCTURED\_BUFFER

Value: 51

The variable is a consume structured buffer.

D3D\_SVT\_MIN8FLOAT

Value: 52

The variable is an 8-byte FLOAT.

D3D\_SVT\_MIN10FLOAT

Value: 53

The variable is a 10-byte FLOAT.

D3D\_SVT\_MIN16FLOAT

Value: 54

The variable is a 16-byte FLOAT.

D3D\_SVT\_MIN12INT

Value: 55

The variable is a 12-byte INT.

D3D\_SVT\_MIN16INT

Value: 56

The variable is a 16-byte INT.

D3D\_SVT\_MIN16UINT

Value: 57

The variable is a 16-byte INT.

D3D10\_SVT\_VOID

The variable is a void pointer.

D3D10\_SVT\_BOOL

The variable is a boolean.

D3D10\_SVT\_INT

The variable is an integer.

D3D10\_SVT\_FLOAT

The variable is a floating-point number.

D3D10\_SVT\_STRING

The variable is a string.

D3D10\_SVT\_TEXTURE

The variable is a texture.

D3D10\_SVT\_TEXTURE1D

The variable is a 1D texture.

**D3D10\_SVT\_TEXTURE2D**

The variable is a 2D texture.

**D3D10\_SVT\_TEXTURE3D**

The variable is a 3D texture.

**D3D10\_SVT\_TEXTURECUBE**

The variable is a texture cube.

**D3D10\_SVT\_SAMPLER**

The variable is a sampler.

**D3D10\_SVT\_SAMPLER1D**

The variable is a 1D sampler.

**D3D10\_SVT\_SAMPLER2D**

The variable is a 2D sampler.

**D3D10\_SVT\_SAMPLER3D**

The variable is a 3D sampler.

**D3D10\_SVT\_SAMPLERCUBE**

The variable is a cube sampler.

**D3D10\_SVT\_PIXELSHADER**

The variable is a pixel shader.

**D3D10\_SVT\_VERTEXSHADER**

The variable is a vertex shader.

**D3D10\_SVT\_PIXELFRAGMENT**

The variable is a pixel fragment.

**D3D10\_SVT\_VERTEXFRAGMENT**

The variable is a vertex fragment.

**D3D10\_SVT\_UINT**

The variable is an unsigned integer.

**D3D10\_SVT\_UINT8**

The variable is an 8-bit unsigned integer.

**D3D10\_SVT\_GEOMETRYSHADER**

The variable is a geometry shader.

**D3D10\_SVT\_RASTERIZER**

The variable is a rasterizer-state object.

**D3D10\_SVT\_DEPTHSTENCIL**

The variable is a depth-stencil-state object.

**D3D10\_SVT\_BLEND**

The variable is a blend-state object.

**D3D10\_SVT\_BUFFER**

The variable is a buffer.

**D3D10\_SVT\_CBUFFER**

The variable is a constant buffer.

**D3D10\_SVT\_TBUFFER**

The variable is a texture buffer.

**D3D10\_SVT\_TEXTURE1DARRAY**

The variable is a 1D-texture array.

**D3D10\_SVT\_TEXTURE2DARRAY**

The variable is a 2D-texture array.

**D3D10\_SVT\_RENDERTARGETVIEW**

The variable is a render-target view.

**D3D10\_SVT\_DEPTHSTENCILVIEW**

The variable is a depth-stencil view.

**D3D10\_SVT\_TEXTURE2DMS**

The variable is a 2D-multisampled texture.

**D3D10\_SVT\_TEXTURE2DMSARRAY**

The variable is a 2D-multisampled-texture array.

**D3D10\_SVT\_TEXTURECUBEARRAY**

The variable is a texture-cube array.

**D3D11\_SVT\_HULLSHADER**

The variable holds a compiled hull-shader binary.

**D3D11\_SVT\_DOMAINSHADER**

The variable holds a compiled domain-shader binary.

**D3D11\_SVT\_INTERFACE\_POINTER**

The variable is an interface.

**D3D11\_SVT\_COMPUTESHADER**

The variable holds a compiled compute-shader binary.

**D3D11\_SVT\_DOUBLE**

The variable is a double precision (64-bit) floating-point number.

**D3D11\_SVT\_RWTEXTURE1D**

The variable is a 1D read-and-write texture.

**D3D11\_SVT\_RWTEXTURE1DARRAY**

The variable is an array of 1D read-and-write textures.

**D3D11\_SVT\_RWTEXTURE2D**

The variable is a 2D read-and-write texture.

**D3D11\_SVT\_RWTEXTURE2DARRAY**

The variable is an array of 2D read-and-write textures.

**D3D11\_SVT\_RWTEXTURE3D**

The variable is a 3D read-and-write texture.

**D3D11\_SVT\_RWBUFFER**

The variable is a read-and-write buffer.

**D3D11\_SVT\_BYTEADDRESS\_BUFFER**

The variable is a byte-address buffer.

**D3D11\_SVT\_RWBYTEADDRESS\_BUFFER**

The variable is a read and write byte-address buffer.

**D3D11\_SVT\_STRUCTURED\_BUFFER**

The variable is a structured buffer.

**D3D11\_SVT\_RWSTRUCTURED\_BUFFER**

The variable is a read-and-write structured buffer.

**D3D11\_SVT\_APPEND\_STRUCTURED\_BUFFER**

The variable is an append structured buffer.

**D3D11\_SVT\_CONSUME\_STRUCTURED\_BUFFER**

The variable is a consume structured buffer.

**D3D\_SVT\_FORCE\_DWORD**

Value: `0x7fffffff`

This value is not used by a programmer; it exists to force the enumeration to compile to 32 bits.

## Remarks

A call to the [ID3D11ShaderReflectionType::GetDesc](#) method returns a **D3D\_SHADER\_VARIABLE\_TYPE** value in the **Type** member of a [D3D11\\_SHADER\\_TYPE\\_DESC](#) structure.

The types in a structured buffer describe the structure of the elements in the buffer. The layout of these types generally match their C++ struct counterparts. The following examples show structured buffers:

```
struct mystruct {float4 val; uint ind;}; RWStructuredBuffer<mystruct> rdbuf;  
RWStructuredBuffer<float3> rdbuf2;
```

## Requirements

 [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Common Version Enumerations](#)

# D3D\_SRV\_DIMENSION enumeration (d3dcommon.h)

Article 01/31/2022

Values that identify the type of resource to be viewed as a shader resource.

## ⓘ Note

For programming with Direct3D 10, this API has a type alias that begins `D3D10_` instead of `D3D_`. These Direct3D 10 type aliases are defined in `d3d10.h`, `d3d10misc.h`, and `d3d10shader.h`.

## Syntax

C++

```
typedef enum D3D_SRV_DIMENSION {
    D3D_SRV_DIMENSION_UNKNOWN = 0,
    D3D_SRV_DIMENSION_BUFFER = 1,
    D3D_SRV_DIMENSION_TEXTURE1D = 2,
    D3D_SRV_DIMENSION_TEXTURE1DARRAY = 3,
    D3D_SRV_DIMENSION_TEXTURE2D = 4,
    D3D_SRV_DIMENSION_TEXTURE2DARRAY = 5,
    D3D_SRV_DIMENSION_TEXTURE2DMS = 6,
    D3D_SRV_DIMENSION_TEXTURE2DMSARRAY = 7,
    D3D_SRV_DIMENSION_TEXTURE3D = 8,
    D3D_SRV_DIMENSION_TEXTURECUBE = 9,
    D3D_SRV_DIMENSION_TEXTURECUBEARRAY = 10,
    D3D_SRV_DIMENSION_BUFFEREX = 11,
    D3D10_SRV_DIMENSION_UNKNOWN,
    D3D10_SRV_DIMENSION_BUFFER,
    D3D10_SRV_DIMENSION_TEXTURE1D,
    D3D10_SRV_DIMENSION_TEXTURE1DARRAY,
    D3D10_SRV_DIMENSION_TEXTURE2D,
    D3D10_SRV_DIMENSION_TEXTURE2DARRAY,
    D3D10_SRV_DIMENSION_TEXTURE2DMS,
    D3D10_SRV_DIMENSION_TEXTURE2DMSARRAY,
    D3D10_SRV_DIMENSION_TEXTURE3D,
    D3D10_SRV_DIMENSION_TEXTURECUBE,
    D3D10_1_SRV_DIMENSION_UNKNOWN,
    D3D10_1_SRV_DIMENSION_BUFFER,
    D3D10_1_SRV_DIMENSION_TEXTURE1D,
    D3D10_1_SRV_DIMENSION_TEXTURE1DARRAY,
    D3D10_1_SRV_DIMENSION_TEXTURE2D,
    D3D10_1_SRV_DIMENSION_TEXTURE2DARRAY,
```

```
D3D10_1_SRV_DIMENSION_TEXTURE2DMS,  
D3D10_1_SRV_DIMENSION_TEXTURE2DMSARRAY,  
D3D10_1_SRV_DIMENSION_TEXTURE3D,  
D3D10_1_SRV_DIMENSION_TEXTURECUBE,  
D3D10_1_SRV_DIMENSION_TEXTURECUBEARRAY,  
D3D11_SRV_DIMENSION_UNKNOWN,  
D3D11_SRV_DIMENSION_BUFFER,  
D3D11_SRV_DIMENSION_TEXTURE1D,  
D3D11_SRV_DIMENSION_TEXTURE1DARRAY,  
D3D11_SRV_DIMENSION_TEXTURE2D,  
D3D11_SRV_DIMENSION_TEXTURE2DARRAY,  
D3D11_SRV_DIMENSION_TEXTURE2DMS,  
D3D11_SRV_DIMENSION_TEXTURE2DMSARRAY,  
D3D11_SRV_DIMENSION_TEXTURE3D,  
D3D11_SRV_DIMENSION_TEXTURECUBE,  
D3D11_SRV_DIMENSION_TEXTURECUBEARRAY,  
D3D11_SRV_DIMENSION_BUFFEREX  
} ;
```

## Constants

`D3D_SRV_DIMENSION_UNKNOWN`

Value: 0

The type is unknown.

`D3D_SRV_DIMENSION_BUFFER`

Value: 1

The resource is a buffer.

`D3D_SRV_DIMENSION_TEXTURE1D`

Value: 2

The resource is a 1D texture.

`D3D_SRV_DIMENSION_TEXTURE1DARRAY`

Value: 3

The resource is an array of 1D textures.

`D3D_SRV_DIMENSION_TEXTURE2D`

Value: 4

The resource is a 2D texture.

`D3D_SRV_DIMENSION_TEXTURE2DARRAY`

Value: 5

The resource is an array of 2D textures.

`D3D_SRV_DIMENSION_TEXTURE2DMS`

Value: 6

The resource is a multisampling 2D texture.

`D3D_SRV_DIMENSION_TEXTURE2DMSARRAY`

Value: 7

The resource is an array of multisampling 2D textures.

`D3D_SRV_DIMENSION_TEXTURE3D`

Value: 8

The resource is a 3D texture.

`D3D_SRV_DIMENSION_TEXTURECUBE`

Value: 9

The resource is a cube texture.

`D3D_SRV_DIMENSION_TEXTURECUBEARRAY`

Value: 10

The resource is an array of cube textures.

`D3D_SRV_DIMENSION_BUFFEREX`

Value: 11

The resource is a raw buffer. For more info about raw viewing of buffers, see [Raw Views of Buffers](#).

`D3D10_SRV_DIMENSION_UNKNOWN`

The type is unknown.

`D3D10_SRV_DIMENSION_BUFFER`

The resource is a buffer.

`D3D10_SRV_DIMENSION_TEXTURE1D`

The resource is a 1D texture.

`D3D10_SRV_DIMENSION_TEXTURE1DARRAY`

The resource is an array of 1D textures.

`D3D10_SRV_DIMENSION_TEXTURE2D`

The resource is a 2D texture.

`D3D10_SRV_DIMENSION_TEXTURE2DARRAY`

The resource is an array of 2D textures.

`D3D10_SRV_DIMENSION_TEXTURE2DMS`

The resource is a multisampling 2D texture.

`D3D10_SRV_DIMENSION_TEXTURE2DMSARRAY`

The resource is an array of multisampling 2D textures.

`D3D10_SRV_DIMENSION_TEXTURE3D`

The resource is a 3D texture.

`D3D10_SRV_DIMENSION_TEXTURECUBE`

The resource is a cube texture.

`D3D10_1_SRV_DIMENSION_UNKNOWN`

The type is unknown.

`D3D10_1_SRV_DIMENSION_BUFFER`

The resource is a buffer.

`D3D10_1_SRV_DIMENSION_TEXTURE1D`

The resource is a 1D texture.

`D3D10_1_SRV_DIMENSION_TEXTURE1DARRAY`

The resource is an array of 1D textures.

`D3D10_1_SRV_DIMENSION_TEXTURE2D`

The resource is a 2D texture.

`D3D10_1_SRV_DIMENSION_TEXTURE2DARRAY`

The resource is an array of 2D textures.

`D3D10_1_SRV_DIMENSION_TEXTURE2DMS`

The resource is a multisampling 2D texture.

`D3D10_1_SRV_DIMENSION_TEXTURE2DMSARRAY`

The resource is an array of multisampling 2D textures.

`D3D10_1_SRV_DIMENSION_TEXTURE3D`

The resource is a 3D texture.

`D3D10_1_SRV_DIMENSION_TEXTURECUBE`

The resource is a cube texture.

`D3D10_1_SRV_DIMENSION_TEXTURECUBEARRAY`

The resource is an array of cube textures.

`D3D11_SRV_DIMENSION_UNKNOWN`

The type is unknown.

`D3D11_SRV_DIMENSION_BUFFER`

The resource is a buffer.

`D3D11_SRV_DIMENSION_TEXTURE1D`

The resource is a 1D texture.

**D3D11\_SRV\_DIMENSION\_TEXTURE1DARRAY**

The resource is an array of 1D textures.

**D3D11\_SRV\_DIMENSION\_TEXTURE2D**

The resource is a 2D texture.

**D3D11\_SRV\_DIMENSION\_TEXTURE2DARRAY**

The resource is an array of 2D textures.

**D3D11\_SRV\_DIMENSION\_TEXTURE2DMS**

The resource is a multisampling 2D texture.

**D3D11\_SRV\_DIMENSION\_TEXTURE2DMSARRAY**

The resource is an array of multisampling 2D textures.

**D3D11\_SRV\_DIMENSION\_TEXTURE3D**

The resource is a 3D texture.

**D3D11\_SRV\_DIMENSION\_TEXTURECUBE**

The resource is a cube texture.

**D3D11\_SRV\_DIMENSION\_TEXTURECUBEARRAY**

The resource is an array of cube textures.

**D3D11\_SRV\_DIMENSION\_BUFFEREX**

The resource is a raw buffer. For more info about raw viewing of buffers, see [Raw Views of Buffers](#).

## Remarks

A D3D\_SRV\_DIMENSION-typed value is specified in the **ViewDimension** member of the [ViewDimension](#) member of the [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure or the **Dimension** member of the [D3D11\\_SHADER\\_INPUT\\_BIND\\_DESC](#) structure.

## Requirements

Header

d3dcommon.h

## See also

[Common Version Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3D\_TESSELLATOR\_DOMAIN enumeration (d3dcommon.h)

Article 02/22/2024

Domain options for tessellator data.

## Syntax

C++

```
typedef enum D3D_TESSELLATOR_DOMAIN {
    D3D_TESSELLATOR_DOMAIN_UNDEFINED = 0,
    D3D_TESSELLATOR_DOMAIN_ISOLINE = 1,
    D3D_TESSELLATOR_DOMAIN_TRI = 2,
    D3D_TESSELLATOR_DOMAIN_QUAD = 3,
    D3D11_TESSELLATOR_DOMAIN_UNDEFINED,
    D3D11_TESSELLATOR_DOMAIN_ISOLINE,
    D3D11_TESSELLATOR_DOMAIN_TRI,
    D3D11_TESSELLATOR_DOMAIN_QUAD
} ;
```

## Constants

[] Expand table

D3D\_TESSELLATOR\_DOMAIN\_UNDEFINED

Value: 0

D3D\_TESSELLATOR\_DOMAIN\_ISOLINE

Value: 1

D3D\_TESSELLATOR\_DOMAIN\_TRI

Value: 2

D3D\_TESSELLATOR\_DOMAIN\_QUAD

Value: 3

D3D11\_TESSELLATOR\_DOMAIN\_UNDEFINED

The data type is undefined.

D3D11\_TESSELLATOR\_DOMAIN\_ISOLINE

Isoline data.

D3D11\_TESSELLATOR\_DOMAIN\_TRI

Triangle data.

D3D11\_TESSELLATOR\_DOMAIN\_QUAD

Quad data.

## Remarks

The data domain defines the type of data. This enumeration is used by [D3D11\\_SHADER\\_DESC](#).

The **D3D11\_TESSELLATOR\_DOMAIN** enumeration is type defined in the D3D11Shader.h header file as a [D3D\\_TESSELLATOR\\_DOMAIN](#) enumeration, which is fully defined in the D3DCCommon.h header file.

```
typedef D3D_TESSELLATOR_DOMAIN D3D11_TESSELLATOR_DOMAIN;
```

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Shader Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_TESSELLATOR\_OUTPUT\_PRIMITIVE enumeration (d3dcommon.h)

Article 02/22/2024

Output primitive types.

## Syntax

C++

```
typedef enum D3D_TESSELLATOR_OUTPUT_PRIMITIVE {
    D3D_TESSELLATOR_OUTPUT_UNDEFINED = 0,
    D3D_TESSELLATOR_OUTPUT_POINT = 1,
    D3D_TESSELLATOR_OUTPUT_LINE = 2,
    D3D_TESSELLATOR_OUTPUT_TRIANGLE_CW = 3,
    D3D_TESSELLATOR_OUTPUT_TRIANGLE_CCW = 4,
    D3D11_TESSELLATOR_OUTPUT_UNDEFINED,
    D3D11_TESSELLATOR_OUTPUT_POINT,
    D3D11_TESSELLATOR_OUTPUT_LINE,
    D3D11_TESSELLATOR_OUTPUT_TRIANGLE_CW,
    D3D11_TESSELLATOR_OUTPUT_TRIANGLE_CCW
} ;
```

## Constants

[+] Expand table

<code>D3D_TESSELLATOR_OUTPUT_UNDEFINED</code>
Value: 0
<code>D3D_TESSELLATOR_OUTPUT_POINT</code>
Value: 1
<code>D3D_TESSELLATOR_OUTPUT_LINE</code>
Value: 2
<code>D3D_TESSELLATOR_OUTPUT_TRIANGLE_CW</code>
Value: 3
<code>D3D_TESSELLATOR_OUTPUT_TRIANGLE_CCW</code>
Value: 4

#### D3D11\_TESSELLATOR\_OUTPUT\_UNDEFINED

The output primitive type is undefined.

#### D3D11\_TESSELLATOR\_OUTPUT\_POINT

The output primitive type is a point.

#### D3D11\_TESSELLATOR\_OUTPUT\_LINE

The output primitive type is a line.

#### D3D11\_TESSELLATOR\_OUTPUT\_TRIANGLE\_CW

The output primitive type is a clockwise triangle.

#### D3D11\_TESSELLATOR\_OUTPUT\_TRIANGLE\_CCW

The output primitive type is a counter clockwise triangle.

## Remarks

The output primitive type determines how the tessellator output data is organized; this enumeration is used by [D3D11\\_SHADER\\_DESC](#).

The **D3D11\_TESSELLATOR\_OUTPUT\_PRIMITIVE** enumeration is type defined in the D3D11Shader.h header file as a [D3D\\_TESSELLATOR\\_OUTPUT\\_PRIMITIVE](#) enumeration, which is fully defined in the D3DCCommon.h header file.

```
typedef D3D_TESSELLATOR_OUTPUT_PRIMITIVE D3D11_TESSELLATOR_OUTPUT_PRIMITIVE;
```

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Shader Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3D\_TESSELLATOR\_PARTITIONING enumeration (d3dcommon.h)

Article 02/22/2024

Partitioning options.

## Syntax

C++

```
typedef enum D3D_TESSELLATOR_PARTITIONING {
    D3D_TESSELLATOR_PARTITIONING_UNDEFINED = 0,
    D3D_TESSELLATOR_PARTITIONING_INTEGER = 1,
    D3D_TESSELLATOR_PARTITIONING_POW2 = 2,
    D3D_TESSELLATOR_PARTITIONING_FRACTIONAL_ODD = 3,
    D3D_TESSELLATOR_PARTITIONING_FRACTIONAL_EVEN = 4,
    D3D11_TESSELLATOR_PARTITIONING_UNDEFINED,
    D3D11_TESSELLATOR_PARTITIONING_INTEGER,
    D3D11_TESSELLATOR_PARTITIONING_POW2,
    D3D11_TESSELLATOR_PARTITIONING_FRACTIONAL_ODD,
    D3D11_TESSELLATOR_PARTITIONING_FRACTIONAL_EVEN
} ;
```

## Constants

[+] Expand table

<code>D3D_TESSELLATOR_PARTITIONING_UNDEFINED</code>
Value: 0
<code>D3D_TESSELLATOR_PARTITIONING_INTEGER</code>
Value: 1
<code>D3D_TESSELLATOR_PARTITIONING_POW2</code>
Value: 2
<code>D3D_TESSELLATOR_PARTITIONING_FRACTIONAL_ODD</code>
Value: 3
<code>D3D_TESSELLATOR_PARTITIONING_FRACTIONAL_EVEN</code>
Value: 4

#### D3D11\_TESSELLATOR\_PARTITIONING\_UNDEFINED

The partitioning type is undefined.

#### D3D11\_TESSELLATOR\_PARTITIONING\_INTEGER

Partition with integers only.

#### D3D11\_TESSELLATOR\_PARTITIONING\_POW2

Partition with a power-of-two number only.

#### D3D11\_TESSELLATOR\_PARTITIONING\_FRACTIONAL\_ODD

Partition with an odd, fractional number.

#### D3D11\_TESSELLATOR\_PARTITIONING\_FRACTIONAL\_EVEN

Partition with an even, fractional number.

## Remarks

During tessellation, the partition option helps to determine how the algorithm chooses the next partition value; this enumeration is used by [D3D11\\_SHADER\\_DESC](#).

The **D3D11\_TESSELLATOR\_PARTITIONING** enumeration is type defined in the D3D11Shader.h header file as a [D3D\\_TESSELLATOR\\_PARTITIONING](#) enumeration, which is fully defined in the D3DCCommon.h header file.

```
typedef D3D_TESSELLATOR_PARTITIONING D3D11_TESSELLATOR_PARTITIONING;
```

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3dcommon.h

## See also

[Shader Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11 Helper Structures

Article • 11/04/2020

Direct3D 11 defines several helper structures that you can use to create Direct3D structures. These helper structures behave like C++ classes.

## In this section

Topic	Description
<a href="#">CD3D11_RECT</a>	Represents a rectangle and provides convenience methods for creating rectangles.
<a href="#">CD3D11_BOX</a>	Represents a box and provides convenience methods for creating boxes.
<a href="#">CD3D11_DEPTH_STENCIL_DESC</a>	Represents a depth-stencil-state structure and provides convenience methods for creating depth-stencil-state structures.
<a href="#">CD3D11_BLEND_DESC</a>	Represents a blend-state structure and provides convenience methods for creating blend-state structures.
<a href="#">CD3D11_RASTERIZER_DESC</a>	Represents a rasterizer-state structure and provides convenience methods for creating rasterizer-state structures.
<a href="#">CD3D11_BUFFER_DESC</a>	Represents a buffer and provides convenience methods for creating buffers.
<a href="#">CD3D11_TEXTURE1D_DESC</a>	Represents a 1D texture and provides convenience methods for creating 1D textures.
<a href="#">CD3D11_TEXTURE2D_DESC</a>	Represents a 2D texture and provides convenience methods for creating 2D textures.
<a href="#">CD3D11_TEXTURE3D_DESC</a>	Represents a 3D texture and provides convenience methods for creating 3D textures.
<a href="#">CD3D11_SHADER_RESOURCE_VIEW_DESC</a>	Represents a shader-resource view and provides convenience methods for creating shader-resource views.
<a href="#">CD3D11_RENDER_TARGET_VIEW_DESC</a>	Represents a render-target view and provides convenience methods for creating render-target views.

Topic	Description
<a href="#">CD3D11_VIEWPORT</a>	Represents a viewport and provides convenience methods for creating viewports.
<a href="#">CD3D11_DEPTH_STENCIL_VIEW_DESC</a>	Represents a depth-stencil view and provides convenience methods for creating depth-stencil views.
<a href="#">CD3D11_UNORDERED_ACCESS_VIEW_DESC</a>	Represents a unordered-access view and provides convenience methods for creating unordered-access views.
<a href="#">CD3D11_SAMPLER_DESC</a>	Represents a sampler state and provides convenience methods for creating sampler states.
<a href="#">CD3D11_QUERY_DESC</a>	Represents a query and provides convenience methods for creating queries.
<a href="#">CD3D11_COUNTER_DESC</a>	Represents a counter and provides convenience methods for creating counters.

## Related topics

[Direct3D 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CD3D11\_RECT structure (d3d11.h)

Article 02/22/2024

Represents a rectangle and provides convenience methods for creating rectangles.

## Syntax

C++

```
struct CD3D11_RECT : D3D11_RECT {
    void CD3D11_RECT();
    void CD3D11_RECT(
        const D3D11_RECT & o
    );
    void CD3D11_RECT(
        LONG Left,
        LONG Top,
        LONG Right,
        LONG Bottom
    );
    void ~CD3D11_RECT();
};
```

## Inheritance

The [CD3D11\\_RECT](#) structure implements [D3D11\\_RECT](#).

## Members

[void CD3D11\\_RECT\(\)](#)

Instantiates a new instance of an uninitialized [CD3D11\\_RECT](#) structure.

[void CD3D11\\_RECT\( const D3D11\\_RECT & o \)](#)

Instantiates a new instance of a [CD3D11\\_RECT](#) structure that is initialized with a [D3D11\\_RECT](#) structure.

[void CD3D11\\_RECT\( LONG Left, LONG Top, LONG Right, LONG Bottom \)](#)

Instantiates a new instance of a [CD3D11\\_RECT](#) structure that is initialized with the dimensions of a rectangle.

[void ~CD3D11\\_RECT\(\)](#)

Destroys an instance of a [CD3D11\\_RECT](#) structure.

## Remarks

Here is how D3D11.h defines [CD3D11\\_RECT](#):

[+] Expand table

```
struct CD3D11_RECT : public D3D11_RECT
{
    CD3D11_RECT()
    {}
    explicit CD3D11_RECT( const D3D11_RECT& o ) :
        D3D11_RECT( o )
    {}
    explicit CD3D11_RECT(
        LONG Left,
        LONG Top,
        LONG Right,
        LONG Bottom )
    {
        left = Left;
        top = Top;
        right = Right;
        bottom = Bottom;
    }
    ~CD3D11_RECT() {}
    operator const D3D11_RECT&() const { return *this; }
};
```

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_RECT::CD3D11\_RECT function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_RECT](#) structure.

## Syntax

C++

```
void CD3D11_RECT();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RECT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_RECT::CD3D11\_RECT function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_RECT](#) structure.

## Syntax

C++

```
void CD3D11_RECT();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RECT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_RECT::CD3D11\_RECT(LONG,LON G,ULONG,ULONG) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RECT](#) structure that is initialized with the dimensions of a rectangle.

## Syntax

C++

```
void CD3D11_RECT(
    LONG Left,
    LONG Top,
    LONG Right,
    LONG Bottom
);
```

## Parameters

Left

Type: **LONG**

The x-coordinate of the upper-left corner of the rectangle.

Top

Type: **LONG**

The y-coordinate of the upper-left corner of the rectangle.

Right

Type: **LONG**

The x-coordinate of the lower-right corner of the rectangle.

Bottom

Type: **LONG**

The y-coordinate of the lower-right corner of the rectangle.

# Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RECT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_RECT::~CD3D11\_RECT function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_RECT](#) structure.

## Syntax

C++

```
void ~CD3D11_RECT();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RECT](#)

# CD3D11\_BOX class

Article 12/01/2017

Represents a box and provides convenience methods for creating boxes.

## Members

The **CD3D11\_BOX** class inherits from [D3D11\\_BOX](#). **CD3D11\_BOX** also has these types of members:

- Constructors
- Methods

## Constructors

The **CD3D11\_BOX** class has these constructors.

[Expand table](#)

Constructor	Description
<a href="#">CD3D11_BOX()</a>	Instantiates a new instance of an uninitialized <b>CD3D11_BOX</b> structure.
<a href="#">CD3D11_BOX(LONG, LONG, LONG)</a>	Instantiates a new instance of a <b>CD3D11_BOX</b> structure that is initialized with the dimensions of a box.

## Methods

The **CD3D11\_BOX** class has these methods.

[Expand table](#)

Method	Description
<a href="#">CD3D11_BOX</a>	Destroys an instance of a <b>CD3D11_BOX</b> structure.
<a href="#">CD3D11_BOX(D3D11_BOX&amp;)(const &amp;CD3D11_BOX)</a>	Instantiates a new instance of a <b>CD3D11_BOX</b> structure that is initialized with a <b>D3D11_BOX</b> structure.

Method	Description
<a href="#">D3D11_BOX()</a>	This operator returns the address of a <b>D3D11_BOX</b> structure that contains the data from the <b>CD3D11_BOX</b> instance.

## Remarks

Here is how D3D11.h defines **CD3D11\_BOX**:

```
struct CD3D11_BOX : public D3D11_BOX
{
    CD3D11_BOX()
    {}
    explicit CD3D11_BOX( const D3D11_BOX& o ) :
        D3D11_BOX( o )
    {}
    explicit CD3D11_BOX(
        LONG Left,
        LONG Top,
        LONG Front,
        LONG Right,
        LONG Bottom,
        LONG Back )
    {
        left = Left;
        top = Top;
        front = Front;
        right = Right;
        bottom = Bottom;
        back = Back;
    }
    ~CD3D11_BOX() {}
    operator const D3D11_BOX&() const { return *this; }
};
```

## Requirements

[\[+\] Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_BOX ↗](#)

[CD3D11 Helper Structures ↗](#)

# CD3D11\_BOX::CD3D11\_BOX function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_BOX](#) structure.

## Syntax

C++

```
void CD3D11_BOX();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BOX](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_BOX::CD3D11\_BOX(const D3D11\_BOX&) function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of a [CD3D11\\_BOX](#) structure that is initialized with a [D3D11\\_BOX](#) structure.

## Syntax

C++

```
void CD3D11_BOX(  
    const D3D11_BOX & o  
) ;
```

## Parameters

o

Instantiates a new instance of a [CD3D11\\_BOX](#) structure that is initialized with a [D3D11\\_BOX](#) structure.

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

Requirement	Value
Library	D3D11.lib

## See also

[CD3D11\\_BOX](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_BOX::CD3D11\_BOX(LONG, LONG, LONG, LONG, LONG, LONG) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_BOX](#) structure that is initialized with the dimensions of a box.

## Syntax

C++

```
void CD3D11_BOX(
    LONG Left,
    LONG Top,
    LONG Front,
    LONG Right,
    LONG Bottom,
    LONG Back
);
```

## Parameters

Left

Type: **LONG**

The x-coordinate of the left hand side of the box.

Top

Type: **LONG**

The y-coordinate of the top of the box.

Front

Type: **LONG**

The z-coordinate of the front of the box.

Right

Type: **LONG**

The x-coordinate of the right hand side of the box.

Bottom

Type: **LONG**

The y-coordinate of the bottom of the box.

Back

Type: **LONG**

The y-coordinate of the back of the box.

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BOX](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_BOX::~CD3D11\_BOX function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_BOX](#) structure.

## Syntax

C++

```
void ~CD3D11_BOX();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BOX](#)

# CD3D11\_DEPTH\_STENCIL\_DESC class

Article 12/01/2017

Represents a depth-stencil-state structure and provides convenience methods for creating depth-stencil-state structures.

## Members

The CD3D11\_DEPTH\_STENCIL\_DESC class inherits from [D3D11\\_DEPTH\\_STENCIL\\_DESC](#). CD3D11\_DEPTH\_STENCIL\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_DEPTH\_STENCIL\_DESC class has these constructors.

[Expand table](#)

Constructor
<a href="#">CD3D11_DEPTH_STENCIL_DESC()</a>
<a href="#">CD3D11_DEPTH_STENCIL_DESC(CD3D11_DEFAULT)(CD3D11_DEFAULT)</a>
<a href="#">CD3D11_DEPTH_STENCIL_DESC(D3D11_DEPTH_STENCIL_DESC_values)</a> (BOOL,D3D11_DEPTH_WRITE_MASK,D3D11_COMPARISON_FUNC,BOOL,UINT8,UINT8,D3D11_STENCIL_OP,D3D11_STENCIL_OP,D3D11_STENCIL_OP,D3D11_COMPARISON_

## Methods

The CD3D11\_DEPTH\_STENCIL\_DESC class has these methods.

[Expand table](#)

Method	Description
<a href="#">~CD3D11_DEPTH_STENCIL_DESC</a>	Destroys an instance of a CD3D11_DEPTH_STENCIL_DESC structure.
<a href="#">CD3D11_DEPTH_STENCIL_DESC(D3D11_DEPTH_STENCIL_DESC&amp;)(const &amp;CD3D11_DEPTH_STENCIL_DESC)</a>	Instantiates a new instance of a CD3D11_DEPTH_STENCIL_DESC structure that is initialized with a D3D11_DEPTH_STENCIL_DESC structure.
<a href="#">D3D11_DEPTH_STENCIL_DESC()</a>	This operator returns the address of a D3D11_DEPTH_STENCIL_DESC structure that contains the data from the CD3D11_DEPTH_STENCIL_DESC instance.

## Remarks

Here is how D3D11.h defines CD3D11\_DEPTH\_STENCIL\_DESC:

```
struct CD3D11_DEPTH_STENCIL_DESC : public D3D11_DEPTH_STENCIL_DESC
{
```

```

CD3D11_DEPTH_STENCIL_DESC()
{}
explicit CD3D11_DEPTH_STENCIL_DESC( const D3D11_DEPTH_STENCIL_DESC& o ) :
    D3D11_DEPTH_STENCIL_DESC( o )
{}
explicit CD3D11_DEPTH_STENCIL_DESC( CD3D11_DEFAULT )
{
    DepthEnable = TRUE;
    DepthWriteMask = D3D11_DEPTH_WRITE_MASK_ALL;
    DepthFunc = D3D11_COMPARISON_LESS;
    StencilEnable = FALSE;
    StencilReadMask = D3D11_DEFAULT_STENCIL_READ_MASK;
    StencilWriteMask = D3D11_DEFAULT_STENCIL_WRITE_MASK;
    const D3D11_DEPTH_STENCILOP_DESC defaultStencilOp =
    { D3D11_STENCIL_OP_KEEP, D3D11_STENCIL_OP_KEEP, D3D11_STENCIL_OP_KEEP, D3D11_COMPARISON_ALWAYS };
    FrontFace = defaultStencilOp;
    BackFace = defaultStencilOp;
}
explicit CD3D11_DEPTH_STENCIL_DESC(
    BOOL depthEnable,
    D3D11_DEPTH_WRITE_MASK depthWriteMask,
    D3D11_COMPARISON_FUNC depthFunc,
    BOOL stencilEnable,
    UINT8 stencilReadMask,
    UINT8 stencilWriteMask,
    D3D11_STENCIL_OP frontStencilFailOp,
    D3D11_STENCIL_OP frontStencilDepthFailOp,
    D3D11_STENCIL_OP frontStencilPassOp,
    D3D11_COMPARISON_FUNC frontStencilFunc,
    D3D11_STENCIL_OP backStencilFailOp,
    D3D11_STENCIL_OP backStencilDepthFailOp,
    D3D11_STENCIL_OP backStencilPassOp,
    D3D11_COMPARISON_FUNC backStencilFunc )
{
    DepthEnable = depthEnable;
    DepthWriteMask = depthWriteMask;
    DepthFunc = depthFunc;
    StencilEnable = stencilEnable;
    StencilReadMask = stencilReadMask;
    StencilWriteMask = stencilWriteMask;
    FrontFace.StencilFailOp = frontStencilFailOp;
    FrontFace.StencilDepthFailOp = frontStencilDepthFailOp;
    FrontFace.StencilPassOp = frontStencilPassOp;
    FrontFace.StencilFunc = frontStencilFunc;
    BackFace.StencilFailOp = backStencilFailOp;
    BackFace.StencilDepthFailOp = backStencilDepthFailOp;
    BackFace.StencilPassOp = backStencilPassOp;
    BackFace.StencilFunc = backStencilFunc;
}
~CD3D11_DEPTH_STENCIL_DESC() {}
operator const D3D11_DEPTH_STENCIL_DESC&() const { return *this; }
};

```

## Requirements

 Expand table

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_DEPTH\\_STENCIL\\_DESC](#)

[CD3D11 Helper Structures](#)

# CD3D11\_DEPTH\_STENCIL\_DESC::CD3D11\_DEPTH\_STENCIL\_DESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_DEPTH\\_STENCIL\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_DEPTH_STENCIL_DESC();
```

## Return value

None

## Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_DESC](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_DEPTH\_STENCIL\_DESC::CD3D11\_DEPTH\_STENCIL\_DESC(const D3D11\_DEPTH\_STENCIL\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_DEPTH\\_STENCIL\\_DESC](#) structure that is initialized with a [D3D11\\_DEPTH\\_STENCIL\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_DEPTH_STENCIL_DESC(  
    const D3D11_DEPTH_STENCIL_DESC & o  
) ;
```

## Parameters

o

Address of the [D3D11\\_DEPTH\\_STENCIL\\_DESC](#) structure that initializes the [D3D11\\_DEPTH\\_STENCIL\\_DESC](#) structure.

## Return value

None

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows

Requirement	Value
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_DEPTH\_STENCIL\_DESC::CD3D11\_DEPTH\_STENCIL\_DESC(CD3D11\_DEFAULT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_DEPTH\\_STENCIL\\_DESC](#) structure that is initialized with default depth-stencil-state values.

## Syntax

C++

```
void CD3D11_DEPTH_STENCIL_DESC(
    CD3D11_DEFAULT unnamedParam1
);
```

## Parameters

unnamedParam1

Default depth-stencil-state values.

## Return value

None

## Remarks

Here are the default depth-stencil-state values for the members of [D3D11\\_DEPTH\\_STENCIL\\_DESC](#):

```
DepthEnable = TRUE;
DepthWriteMask = D3D11_DEPTH_WRITE_MASK_ALL;
DepthFunc = D3D11_COMPARISON_LESS;
StencilEnable = FALSE;
StencilReadMask = D3D11_DEFAULT_STENCIL_READ_MASK;
StencilWriteMask = D3D11_DEFAULT_STENCIL_WRITE_MASK;
```

```
const D3D11_DEPTH_STENCILOP_DESC defaultStencilOp =
{ D3D11_STENCIL_OP_KEEP, D3D11_STENCIL_OP_KEEP,
D3D11_STENCIL_OP_KEEP, D3D11_COMPARISON_ALWAYS };
    FrontFace = defaultStencilOp;
    BackFace = defaultStencilOp;
```

# Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_DESC](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_DEPTH\_STENCIL\_DESC::~CD3D11\_DEPTH\_STENCIL\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_DEPTH\\_STENCIL\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_DEPTH_STENCIL_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_DESC](#)

# CD3D11\_BLEND\_DESC structure (d3d11.h)

Article 02/22/2024

Represents a blend-state structure and provides convenience methods for creating blend-state structures.

## Syntax

C++

```
struct CD3D11_BLEND_DESC : D3D11_BLEND_DESC {
    void CD3D11_BLEND_DESC();
    void CD3D11_BLEND_DESC(
        const D3D11_BLEND_DESC & o
    );
    void CD3D11_BLEND_DESC(
        CD3D11_DEFAULT unnamedParam1
    );
    void ~CD3D11_BLEND_DESC();
};
```

## Inheritance

The `CD3D11_BLEND_DESC` structure implements `D3D11_BLEND_DESC`.

## Members

`void CD3D11_BLEND_DESC()`

Instantiates a new instance of an uninitialized `CD3D11_BLEND_DESC` structure.

`void CD3D11_BLEND_DESC( const D3D11_BLEND_DESC & o)`

Instantiates a new instance of a `CD3D11_BLEND_DESC` structure that is initialized with a `CD3D11_BLEND_DESC` structure.

`void CD3D11_BLEND_DESC( CD3D11_DEFAULT unnamedParam1)`

Instantiates a new instance of a `CD3D11_BLEND_DESC` structure that is initialized with default blend-state values.

```
void ~CD3D11_BLEND_DESC()
```

Destroys an instance of a [CD3D11\\_BLEND\\_DESC](#) structure.

## Remarks

Here is how D3D11.h defines [CD3D11\\_BLEND\\_DESC](#):

```
struct CD3D11_BLEND_DESC : public D3D11_BLEND_DESC
{
    CD3D11_BLEND_DESC()
    {}
    explicit CD3D11_BLEND_DESC( const D3D11_BLEND_DESC& o ) :
        D3D11_BLEND_DESC( o )
    {}
    explicit CD3D11_BLEND_DESC( CD3D11_DEFAULT )
    {
        AlphaToCoverageEnable = FALSE;
        IndependentBlendEnable = FALSE;
        const D3D11_RENDER_TARGET_BLEND_DESC defaultRenderTargetBlendDesc =
        {
            FALSE,
            D3D11_BLEND_ONE, D3D11_BLEND_ZERO, D3D11_BLEND_OP_ADD,
            D3D11_BLEND_ONE, D3D11_BLEND_ZERO, D3D11_BLEND_OP_ADD,
            D3D11_COLOR_WRITE_ENABLE_ALL,
        };
        for (UINT i = 0; i < D3D11_SIMULTANEOUS_RENDER_TARGET_COUNT; ++i)
            RenderTarget[ i ] = defaultRenderTargetBlendDesc;
    }
    ~CD3D11_BLEND_DESC() {}
    operator const D3D11_BLEND_DESC&() const { return *this; }
};
```

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	d3d11.h

## See also

[CD3D11 Helper Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_BLEND\_DESC::CD3D11\_BLEND\_D ESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_BLEND\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_BLEND_DESC();
```

## Return value

None

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BLEND\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_BLEND\_DESC::CD3D11\_BLEND\_D ESC(const D3D11\_BLEND\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_BLEND\\_DESC](#) structure that is initialized with a **CD3D11\_BLEND\_DESC** structure.

## Syntax

C++

```
void CD3D11_BLEND_DESC(  
    const D3D11_BLEND_DESC & o  
) ;
```

## Parameters

o

Address of the **CD3D11\_BLEND\_DESC** structure that initializes the **CD3D11\_BLEND\_DESC** structure.

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows

Requirement	Value
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BLEND\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_BLEND\_DESC::CD3D11\_BLEND\_D ESC(CD3D11\_DEFAULT) function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of a [CD3D11\\_BLEND\\_DESC](#) structure that is initialized with default blend-state values.

## Syntax

C++

```
void CD3D11_BLEND_DESC(  
    CD3D11_DEFAULT unnamedParam1  
) ;
```

## Parameters

unnamedParam1

TBD

## Return value

None

## Remarks

Here are the default depth-stencil-state values for the members of [D3D11\\_BLEND\\_DESC](#):

```
AlphaToCoverageEnable = FALSE;  
IndependentBlendEnable = FALSE;  
const D3D11_RENDER_TARGET_BLEND_DESC defaultRenderTargetBlendDesc =  
{  
    FALSE,  
    D3D11_BLEND_ONE, D3D11_BLEND_ZERO, D3D11_BLEND_OP_ADD,  
    D3D11_BLEND_ONE, D3D11_BLEND_ZERO, D3D11_BLEND_OP_ADD,  
    D3D11_COLOR_WRITE_ENABLE_ALL,
```

```
};  
for (UINT i = 0; i < D3D11_SIMULTANEOUS_RENDER_TARGET_COUNT; ++i)  
    RenderTarget[ i ] = defaultRenderTargetBlendDesc;
```

# Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BLEND\\_DESC](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_BLEND\_DESC::~CD3D11\_BLEND\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_BLEND\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_BLEND_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BLEND\\_DESC](#)

# D3D11\_BLEND\_DESC structure (d3d11.h)

Article 02/22/2024

Describes the blend state that you use in a call to [ID3D11Device::CreateBlendState](#) to create a blend-state object.

## Syntax

C++

```
typedef struct D3D11_BLEND_DESC {
    BOOL           AlphaToCoverageEnable;
    BOOL           IndependentBlendEnable;
    D3D11_RENDER_TARGET_BLEND_DESC RenderTarget[8];
} D3D11_BLEND_DESC;
```

## Members

`AlphaToCoverageEnable`

Type: [BOOL](#)

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`

Type: [BOOL](#)

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.

`RenderTarget[8]`

Type: [D3D11\\_RENDER\\_TARGET\\_BLEND\\_DESC\[8\]](#)

An array of [D3D11\\_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

Here are the default values for blend state.

[+] Expand table

State	Default Value
AlphaToCoverageEnable	FALSE
IndependentBlendEnable	FALSE
RenderTarget[0].BlendEnable	FALSE
RenderTarget[0].SrcBlend	D3D11_BLEND_ONE
RenderTarget[0].DestBlend	D3D11_BLEND_ZERO
RenderTarget[0].BlendOp	D3D11_BLEND_OP_ADD
RenderTarget[0].SrcBlendAlpha	D3D11_BLEND_ONE
RenderTarget[0].DestBlendAlpha	D3D11_BLEND_ZERO
RenderTarget[0].BlendOpAlpha	D3D11_BLEND_OP_ADD
RenderTarget[0].RenderTargetWriteMask	D3D11_COLOR_WRITE_ENABLE_ALL

**Note** `D3D11_BLEND_DESC` is identical to [D3D10\\_BLEND\\_DESC1](#).

If the driver type is set to [D3D\\_DRIVER\\_TYPE\\_HARDWARE](#), the feature level is set to less than or equal to [D3D\\_FEATURE\\_LEVEL\\_9\\_3](#), and the pixel format of the render target is set to [DXGI\\_FORMAT\\_R8G8B8A8\\_UNORM\\_SRGB](#), [DXGI\\_FORMAT\\_B8G8R8A8\\_UNORM\\_SRGB](#), or [DXGI\\_FORMAT\\_B8G8R8X8\\_UNORM\\_SRGB](#), the display device performs the blend in standard RGB (sRGB) space and not in linear space. However, if the feature level is set to greater than [D3D\\_FEATURE\\_LEVEL\\_9\\_3](#), the display device performs the blend in linear space, which is ideal.

# Requirements

[+] Expand table

Requirement	Value
Header	d3d11.h

## See also

[Core Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_RASTERIZER\_DESC class

Article 12/01/2017

Represents a rasterizer-state structure and provides convenience methods for creating rasterizer-state structures.

## Members

The CD3D11\_RASTERIZER\_DESC class inherits from [D3D11\\_RASTERIZER\\_DESC](#). CD3D11\_RASTERIZER\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_RASTERIZER\_DESC class has these constructors.

[Expand table](#)

Constructor	Description
<a href="#">CD3D11_RASTERIZER_DESC</a>	Instantiates a new instance of an uninitialized CD3D11_RASTERIZER_DESC structure.
<a href="#">CD3D11_RASTERIZER_DESC(CD3D11_DEFAULT)(CD3D11_DEFAULT)</a>	Instantiates a new instance of a CD3D11_RASTERIZER_DESC structure that is initialized with default rasterizer-state values.
<a href="#">CD3D11_RASTERIZER_DESC(D3D11_FILL_MODE,D3D11_CULL_MODE,BOOL,INT,FLOAT,FLOAT,BOOL,BOOL,BOOL,BOOL)</a> <a href="#">(D3D11_FILL_MODE,D3D11_CULL_MODE,BOOL,INT,FLOAT,FLOAT,BOOL,BOOL,BOOL)</a>	Instantiates a new instance of a CD3D11_RASTERIZER_DESC structure that is initialized with <a href="#">D3D11_RASTERIZER_DESC</a> values.

## Methods

The CD3D11\_RASTERIZER\_DESC class has these methods.

[Expand table](#)

Method	Description
<a href="#">~CD3D11_RASTERIZER_DESC()</a>	Destroys an instance of a CD3D11_RASTERIZER_DESC structure.
<a href="#">CD3D11_RASTERIZER_DESC(const &amp;D3D11_RASTERIZER_DESC)</a>	Instantiates a new instance of a CD3D11_RASTERIZER_DESC structure that is initialized with a D3D11_RASTERIZER_DESC structure.
<a href="#">D3D11_RASTERIZER_DESC()</a>	This operator returns the address of a D3D11_RASTERIZER_DESC structure that contains the data from the CD3D11_RASTERIZER_DESC instance.

## Remarks

Here is how D3D11.h defines **CD3D11\_RASTERIZER\_DESC**:

```
struct CD3D11_RASTERIZER_DESC : public D3D11_RASTERIZER_DESC
{
    CD3D11_RASTERIZER_DESC()
    {}
    explicit CD3D11_RASTERIZER_DESC( const D3D11_RASTERIZER_DESC& o ) :
        D3D11_RASTERIZER_DESC( o )
    {}
    explicit CD3D11_RASTERIZER_DESC( CD3D11_DEFAULT )
    {
        FillMode = D3D11_FILL_SOLID;
        CullMode = D3D11_CULL_BACK;
        FrontCounterClockwise = FALSE;
        DepthBias = D3D11_DEFAULT_DEPTH_BIAS;
        DepthBiasClamp = D3D11_DEFAULT_DEPTH_BIAS_CLAMP;
        SlopeScaledDepthBias = D3D11_DEFAULT_SLOPE_SCALED_DEPTH_BIAS;
        DepthClipEnable = TRUE;
        ScissorEnable = FALSE;
        MultisampleEnable = FALSE;
        AntialiasedLineEnable = FALSE;
    }
    explicit CD3D11_RASTERIZER_DESC(
        D3D11_FILL_MODE fillMode,
        D3D11_CULL_MODE cullMode,
        BOOL frontCounterClockwise,
        INT depthBias,
        FLOAT depthBiasClamp,
        FLOAT slopeScaledDepthBias,
        BOOL depthClipEnable,
        BOOL scissorEnable,
        BOOL multisampleEnable,
        BOOL antialiasedLineEnable )
    {
        FillMode = fillMode;
        CullMode = cullMode;
        FrontCounterClockwise = frontCounterClockwise;
        DepthBias = depthBias;
        DepthBiasClamp = depthBiasClamp;
        SlopeScaledDepthBias = slopeScaledDepthBias;
        DepthClipEnable = depthClipEnable;
        ScissorEnable = scissorEnable;
        MultisampleEnable = multisampleEnable;
        AntialiasedLineEnable = antialiasedLineEnable;
    }
    ~CD3D11_RASTERIZER_DESC() {}
    operator const D3D11_RASTERIZER_DESC&() const { return *this; }
};
```

## Requirements

[Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_RASTERIZER\\_DESC](#) ↗

[CD3D11 Helper Structures](#) ↗

# CD3D11\_RASTERIZER\_DESC::CD3D11\_RASTERIZER\_DESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_RASTERIZER\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_RASTERIZER_DESC();
```

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RASTERIZER\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_RASTERIZER\_DESC::CD3D11\_RASTERIZER\_DESC(const D3D11\_RASTERIZER\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RASTERIZER\\_DESC](#) structure that is initialized with a [D3D11\\_RASTERIZER\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_RASTERIZER_DESC(  
    const D3D11_RASTERIZER_DESC & o  
) ;
```

## Parameters

o

Address of the [D3D11\\_RASTERIZER\\_DESC](#) structure that initializes the [D3D11\\_RASTERIZER\\_DESC](#) structure.

## Return value

None

## Remarks

Here are the default rasterizer-state values for the members of [D3D11\\_RASTERIZER\\_DESC](#):

```
FillMode = D3D11_FILL_SOLID;  
CullMode = D3D11_CULL_BACK;  
FrontCounterClockwise = FALSE;  
DepthBias = D3D11_DEFAULT_DEPTH_BIAS;  
DepthBiasClamp = D3D11_DEFAULT_DEPTH_BIAS_CLAMP;
```

```
SlopeScaledDepthBias = D3D11_DEFAULT_SLOPE_SCALED_DEPTH_BIAS;  
DepthClipEnable = TRUE;  
ScissorEnable = FALSE;  
MultisampleEnable = FALSE;  
AntialiasedLineEnable = FALSE;
```

# Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RASTERIZER\\_DESC](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_RASTERIZER\_DESC::CD3D11\_RASTERIZER\_DESC(CD3D11\_DEFAULT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RASTERIZER\\_DESC](#) structure that is initialized with default rasterizer-state values.

## Syntax

C++

```
void CD3D11_RASTERIZER_DESC(  
    CD3D11_DEFAULT unnamedParam1  
) ;
```

## Parameters

unnamedParam1

Address of the [D3D11\\_RASTERIZER\\_DESC](#) structure that initializes the [CD3D11\\_RASTERIZER\\_DESC](#) structure.

## Return value

None

## Remarks

Here are the default rasterizer-state values for the members of [D3D11\\_RASTERIZER\\_DESC](#):

```
FillMode = D3D11_FILL_SOLID;  
CullMode = D3D11_CULL_BACK;  
FrontCounterClockwise = FALSE;  
DepthBias = D3D11_DEFAULT_DEPTH_BIAS;  
DepthBiasClamp = D3D11_DEFAULT_DEPTH_BIAS_CLAMP;
```

```
SlopeScaledDepthBias = D3D11_DEFAULT_SLOPE_SCALED_DEPTH_BIAS;  
DepthClipEnable = TRUE;  
ScissorEnable = FALSE;  
MultisampleEnable = FALSE;  
AntialiasedLineEnable = FALSE;
```

# Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RASTERIZER\\_DESC](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_RASTERIZER\_DESC::CD3D11\_RASTERIZER\_DESC(D3D11\_FILL\_MODE,D3D11\_CULL\_MODE,BOOL,INT,FLOAT,FLOAT,BOOL,BOOL,BOOL) function (d3d11.h)

Article10/06/2021

Instantiates a new instance of a [CD3D11\\_RASTERIZER\\_DESC](#) structure that is initialized with [D3D11\\_RASTERIZER\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_RASTERIZER_DESC(
    D3D11_FILL_MODE fillMode,
    D3D11_CULL_MODE cullMode,
    BOOL frontCounterClockwise,
    INT depthBias,
    FLOAT depthBiasClamp,
    FLOAT slopeScaledDepthBias,
    BOOL depthClipEnable,
    BOOL scissorEnable,
    BOOL multisampleEnable,
    BOOL antialiasedLineEnable
);
```

## Parameters

`fillMode`

Type: [D3D11\\_FILL\\_MODE](#)

A [D3D11\\_FILL\\_MODE](#)-typed value that determines the fill mode to use when rendering.

`cullMode`

Type: [D3D11\\_CULL\\_MODE](#)

A [D3D11\\_CULL\\_MODE](#)-typed value that indicates triangles facing the specified direction are not drawn.

`frontCounterClockwise`

Type: [BOOL](#)

A Boolean value that specifies whether a triangle is front- or back-facing. If this parameter 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`

Type: [INT](#)

Depth value added to a given pixel. For info about depth bias, see [Depth Bias](#).

`depthBiasClamp`

Type: [FLOAT](#)

Maximum depth bias of a pixel. For info about depth bias, see [Depth Bias](#).

`slopeScaledDepthBias`

Type: [FLOAT](#)

Scalar on a given pixel's slope. For info about depth bias, see [Depth Bias](#).

`depthClipEnable`

Type: [BOOL](#)

A Boolean value that specifies whether to enable clipping based on distance.

The hardware always performs x and y clipping of rasterized coordinates. When `depthClipEnable` is set to the default—**TRUE**, the hardware also clips the z value (that is, the hardware performs the last step of the following algorithm).

`syntax`

```
0 < w  
-w <= x <= w (or arbitrarily wider range if implementation uses a  
guard band to reduce clipping burden)  
-w <= y <= w (or arbitrarily wider range if implementation uses a  
guard band to reduce clipping burden)
```

```
0 < z < w
```

When you set *depthClipEnable* to **FALSE**, the hardware skips the z clipping (that is, the last step in the preceding algorithm). However, the hardware still performs the "0 < w" clipping. When z clipping is disabled, improper depth ordering at the pixel level might result. However, when z clipping is disabled, stencil shadow implementations are simplified. In other words, you can avoid complex special-case handling for geometry that goes beyond the back clipping plane.

`scissorEnable`

Type: **BOOL**

A Boolean value that specifies whether to enable scissor-rectangle culling. All pixels outside an active scissor rectangle are culled.

`multisampleEnable`

Type: **BOOL**

A Boolean value that 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.

`antialiasedLineEnable`

Type: **BOOL**

A Boolean value that specifies whether to enable line antialiasing; only applies if doing line drawing and *multisampleEnable* is **FALSE**.

## Return value

None

## Remarks

Here is how CD3D11\_RASTERIZER\_DESC assigns the provided values to the members of [D3D11\\_RASTERIZER\\_DESC](#):

```
FillMode = fillMode;
CullMode = cullMode;
FrontCounterClockwise = frontCounterClockwise;
DepthBias = depthBias;
DepthBiasClamp = depthBiasClamp;
SlopeScaledDepthBias = slopeScaledDepthBias;
DepthClipEnable = depthClipEnable;
ScissorEnable = scissorEnable;
MultisampleEnable = multisampleEnable;
AntialiasedLineEnable = antialiasedLineEnable;
```

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RASTERIZER\\_DESC](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CD3D11\_RASTERIZER\_DESC::~CD3D11\_RASTERIZER\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_RASTERIZER\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_RASTERIZER_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RASTERIZER\\_DESC](#)

# CD3D11\_BUFFER\_DESC structure (d3d11.h)

Article 02/16/2023

Represents a buffer and provides convenience methods for creating buffers.

## Syntax

C++

```
struct CD3D11_BUFFER_DESC : D3D11_BUFFER_DESC {
    void CD3D11_BUFFER_DESC();
    void CD3D11_BUFFER_DESC(
        const D3D11_BUFFER_DESC & o
    );
    void CD3D11_BUFFER_DESC(
        UINT byteWidth,
        UINT bindFlags,
        D3D11_USAGE usage,
        UINT cpuaccessFlags,
        UINT miscFlags,
        UINT structureByteStride
    );
    void ~CD3D11_BUFFER_DESC();
};
```

## Inheritance

The `CD3D11_BUFFER_DESC` structure implements `D3D11_BUFFER_DESC`.

## Members

`void CD3D11_BUFFER_DESC()`

Instantiates a new instance of an uninitialized `CD3D11_BUFFER_DESC` structure.

`void CD3D11_BUFFER_DESC( const D3D11_BUFFER_DESC & o )`

Instantiates a new instance of a `CD3D11_BUFFER_DESC` structure that is initialized with a `D3D11_BUFFER_DESC` structure.

`void CD3D11_BUFFER_DESC( UINT byteWidth, UINT bindFlags, D3D11_USAGE usage,
 UINT cpuaccessFlags, UINT miscFlags, UINT structureByteStride )`

Instantiates a new instance of a [CD3D11\\_BUFFER\\_DESC](#) structure that is initialized with [D3D11\\_BUFFER\\_DESC](#) values.

```
void ~CD3D11_BUFFER_DESC()
```

Destroys an instance of a [CD3D11\\_BUFFER\\_DESC](#) structure.

## Remarks

Here is how D3D11.h defines [CD3D11\\_BUFFER\\_DESC](#):

```
struct CD3D11_BUFFER_DESC : public D3D11_BUFFER_DESC
{
    CD3D11_BUFFER_DESC()
    {}
    explicit CD3D11_BUFFER_DESC( const D3D11_BUFFER_DESC& o ) :
        D3D11_BUFFER_DESC( o )
    {}
    explicit CD3D11_BUFFER_DESC(
        UINT byteWidth,
        UINT bindFlags,
        D3D11_USAGE usage = D3D11_USAGE_DEFAULT,
        UINT cpuaccessFlags = 0,
        UINT miscFlags = 0,
        UINT structureByteStride = 0 )
    {
        ByteWidth = byteWidth;
        Usage = usage;
        BindFlags = bindFlags;
        CPUAccessFlags = cpuaccessFlags ;
        MiscFlags = miscFlags;
        StructureByteStride = structureByteStride;
    }
    ~CD3D11_BUFFER_DESC() {}
    operator const D3D11_BUFFER_DESC&() const { return *this; }
};
```

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

## See also

[CD3D11 Helper Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CD3D11\_BUFFER\_DESC::CD3D11\_BUFFER\_DESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_BUFFER\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_BUFFER_DESC();
```

## Return value

None

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BUFFER\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_BUFFER\_DESC::CD3D11\_BUFFER\_DESC(const D3D11\_BUFFER\_DESC&) function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of a [CD3D11\\_BUFFER\\_DESC](#) structure that is initialized with a [D3D11\\_BUFFER\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_BUFFER_DESC(  
    [ref] const D3D11_BUFFER_DESC & o  
) ;
```

## Parameters

[ref] o

Type: [const D3D11\\_BUFFER\\_DESC](#)

Address of the [D3D11\\_BUFFER\\_DESC](#) structure that initializes the [CD3D11\\_BUFFER\\_DESC](#) structure.

## Return value

None

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BUFFER\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_BUFFER\_DESC::CD3D11\_BUFFER\_DESC(UINT,UINT,D3D11\_USAGE,UINT,UINT,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_BUFFER\\_DESC](#) structure that is initialized with [D3D11\\_BUFFER\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_BUFFER_DESC(
    UINT          byteWidth,
    UINT          bindFlags,
    D3D11_USAGE   usage,
    UINT          cpuAccessFlags,
    UINT          miscFlags,
    UINT          structureByteStride
);
```

## Parameters

`byteWidth`

Type: [UINT](#)

Size of the buffer in bytes.

`bindFlags`

Type: [UINT](#)

A combination of [D3D11\\_BIND\\_FLAG](#) values that are combined by using a bitwise OR operation. The resulting value identifies how the buffer will be bound to the pipeline.

`usage`

Type: [D3D11\\_USAGE](#)

A [D3D11\\_USAGE](#)-typed value that identifies how the buffer is expected to be read from and written to. Frequency of update is a key factor.

`cpuaccessFlags`

Type: [UINT](#)

A combination of [D3D11\\_CPU\\_ACCESS\\_FLAG](#) values that are combined by using a bitwise OR operation or 0 if no CPU access is necessary. The resulting value identifies CPU access.

`miscFlags`

Type: [UINT](#)

A combination of [D3D11\\_RESOURCE\\_MISC\\_FLAG](#) values that are combined by using a bitwise OR operation or 0 if unused. The resulting value identifies miscellaneous buffer info.

`structureByteStride`

Type: [UINT](#)

The size of each element in the buffer structure (in bytes) when the buffer represents a structured buffer. For more info about structured buffers, see [Structured Buffer](#).

The size value in *structureByteStride* must match the size of the format that you use for views of the buffer. For example, if you use a shader resource view (SRV) to read a buffer in a pixel shader, the SRV format size must match the size value in *structureByteStride*.

## Return value

None

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

Requirement	Value
Library	D3D11.lib

## See also

[CD3D11\\_BUFFER\\_DESC](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_BUFFER\_DESC::~CD3D11\_BUFFER\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_BUFFER\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_BUFFER_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_BUFFER\\_DESC](#)

# CD3D11\_BUFFER\_DESC structure (d3d11.h)

Article 02/16/2023

Represents a buffer and provides convenience methods for creating buffers.

## Syntax

C++

```
struct CD3D11_BUFFER_DESC : D3D11_BUFFER_DESC {
    void CD3D11_BUFFER_DESC();
    void CD3D11_BUFFER_DESC(
        const D3D11_BUFFER_DESC & o
    );
    void CD3D11_BUFFER_DESC(
        UINT byteWidth,
        UINT bindFlags,
        D3D11_USAGE usage,
        UINT cpuaccessFlags,
        UINT miscFlags,
        UINT structureByteStride
    );
    void ~CD3D11_BUFFER_DESC();
};
```

## Inheritance

The `CD3D11_BUFFER_DESC` structure implements `D3D11_BUFFER_DESC`.

## Members

`void CD3D11_BUFFER_DESC()`

Instantiates a new instance of an uninitialized `CD3D11_BUFFER_DESC` structure.

`void CD3D11_BUFFER_DESC( const D3D11_BUFFER_DESC & o )`

Instantiates a new instance of a `CD3D11_BUFFER_DESC` structure that is initialized with a `D3D11_BUFFER_DESC` structure.

`void CD3D11_BUFFER_DESC( UINT byteWidth, UINT bindFlags, D3D11_USAGE usage,
 UINT cpuaccessFlags, UINT miscFlags, UINT structureByteStride )`

Instantiates a new instance of a [CD3D11\\_BUFFER\\_DESC](#) structure that is initialized with [D3D11\\_BUFFER\\_DESC](#) values.

```
void ~CD3D11_BUFFER_DESC()
```

Destroys an instance of a [CD3D11\\_BUFFER\\_DESC](#) structure.

## Remarks

Here is how D3D11.h defines [CD3D11\\_BUFFER\\_DESC](#):

```
struct CD3D11_BUFFER_DESC : public D3D11_BUFFER_DESC
{
    CD3D11_BUFFER_DESC()
    {}
    explicit CD3D11_BUFFER_DESC( const D3D11_BUFFER_DESC& o ) :
        D3D11_BUFFER_DESC( o )
    {}
    explicit CD3D11_BUFFER_DESC(
        UINT byteWidth,
        UINT bindFlags,
        D3D11_USAGE usage = D3D11_USAGE_DEFAULT,
        UINT cpuaccessFlags = 0,
        UINT miscFlags = 0,
        UINT structureByteStride = 0 )
    {
        ByteWidth = byteWidth;
        Usage = usage;
        BindFlags = bindFlags;
        CPUAccessFlags = cpuaccessFlags ;
        MiscFlags = miscFlags;
        StructureByteStride = structureByteStride;
    }
    ~CD3D11_BUFFER_DESC() {}
    operator const D3D11_BUFFER_DESC&() const { return *this; }
};
```

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

## See also

[CD3D11 Helper Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CD3D11\_TEXTURE1D\_DESC class

Article 12/01/2017

Represents a 1D texture and provides convenience methods for creating 1D textures.

## Members

The CD3D11\_TEXTURE1D\_DESC class inherits from [D3D11\\_TEXTURE1D\\_DESC](#).

CD3D11\_TEXTURE1D\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_TEXTURE1D\_DESC class has these constructors.

[Expand table](#)

Constructor	Description
<a href="#">CD3D11_TEXTURE1D_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_TEXTURE1D_DESC structure.
<a href="#">CD3D11_TEXTURE1D_DESC(D3D11_TEXTURE1D_DESC_values)</a> <a href="#">(DXGI_FORMAT,UINT,UINT,UINT,UINT,D3D11_USAGE,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_TEXTURE1D_DESC structure that is initialized with D3D11_TEXTURE1D_DESC values.

## Methods

The CD3D11\_TEXTURE1D\_DESC class has these methods.

[Expand table](#)

Method	Description
<a href="#">~CD3D11_TEXTURE1D_DESC</a>	Destroys an instance of a CD3D11_TEXTURE1D_DESC structure.

Method	Description
<a href="#">CD3D11_TEXTURE1D_DESC(D3D11_TEXTURE1D_DESC&amp;)</a> <a href="#">(const &amp;D3D11_TEXTURE1D_DESC)</a>	Instantiates a new instance of a <a href="#">CD3D11_TEXTURE1D_DESC</a> structure that is initialized with a <a href="#">D3D11_TEXTURE1D_DESC</a> structure.
<a href="#">D3D11_TEXTURE1D_DESC()</a>	This operator returns the address of a <a href="#">D3D11_TEXTURE1D_DESC</a> structure that contains the data from the <a href="#">CD3D11_TEXTURE1D_DESC</a> instance.

## Remarks

Here is how D3D11.h defines [CD3D11\\_TEXTURE1D\\_DESC](#):

```
struct CD3D11_TEXTURE1D_DESC : public D3D11_TEXTURE1D_DESC
{
    CD3D11_TEXTURE1D_DESC()
    {}
    explicit CD3D11_TEXTURE1D_DESC( const D3D11_TEXTURE1D_DESC& o ) :
        D3D11_TEXTURE1D_DESC( o )
    {}
    explicit CD3D11_TEXTURE1D_DESC(
        DXGI_FORMAT format,
        UINT width,
        UINT arraySize = 1,
        UINT mipLevels = 0,
        UINT bindFlags = D3D11_BIND_SHADER_RESOURCE,
        D3D11_USAGE usage = D3D11_USAGE_DEFAULT,
        UINT cpuaccessFlags= 0,
        UINT miscFlags = 0 )
    {
        Width = width;
        MipLevels = mipLevels;
        ArraySize = arraySize;
        Format = format;
        Usage = usage;
        BindFlags = bindFlags;
        CPUAccessFlags = cpuaccessFlags;
        MiscFlags = miscFlags;
    }
    ~CD3D11_TEXTURE1D_DESC() {}
    operator const D3D11_TEXTURE1D_DESC&() const { return *this; }
};
```

# Requirements

 [Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_TEXTURE1D\\_DESC](#) ↗

[CD3D11 Helper Structures](#) ↗

# CD3D11\_TEXTURE1D\_DESC::CD3D11\_TEXTURE1D\_DESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_TEXTURE1D\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_TEXTURE1D_DESC();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE1D\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_TEXTURE1D\_DESC::CD3D11\_TEXTURE1D\_DESC(const D3D11\_TEXTURE1D\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_TEXTURE1D\\_DESC](#) structure that is initialized with a [D3D11\\_TEXTURE1D\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_TEXTURE1D_DESC(
    [ref] const D3D11_TEXTURE1D_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_TEXTURE1D\\_DESC](#)

Address of the [D3D11\\_TEXTURE1D\\_DESC](#) structure that initializes the [CD3D11\\_TEXTURE1D\\_DESC](#) structure.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE1D\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# `CD3D11_TEXTURE1D_DESC::CD3D11_TEXTURE1D_DESC(DXGI_FORMAT,UINT,UINT,UINT,UINT,D3D11_USAGE,UINT,UINT)` function (d3d11.h)

Article04/02/2021

Instantiates a new instance of a `CD3D11_TEXTURE1D_DESC` structure that is initialized with `D3D11_TEXTURE1D_DESC` values.

## Syntax

C++

```
void CD3D11_TEXTURE1D_DESC(
    DXGI_FORMAT format,
    UINT width,
    UINT arraySize,
    UINT mipLevels,
    UINT bindFlags,
    D3D11_USAGE usage,
    UINT cpuAccessFlags,
    UINT miscFlags
);
```

## Parameters

`format`

Type: `DXGI_FORMAT`

A `DXGI_FORMAT`-typed value that specifies the texture format.

`width`

Type: `UINT`

Texture width (in texels). The range is from 1 to `D3D11_REQ_TEXTURE1D_U_DIMENSION` (16384). However, the range is actually constrained by the `feature level` at which you create the rendering device.

`arraySize`

Type: [UINT](#)

Number of textures in the array. The range is from 1 to [D3D11\\_REQ\\_TEXTURE1D\\_ARRAY\\_AXIS\\_DIMENSION](#) (2048). However, the range is actually constrained by the [feature level](#) at which you create the rendering device.

`mipLevels`

Type: [UINT](#)

The maximum number of mipmap levels in the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#). Use 1 for a multisampled texture; or 0 to generate a full set of subtextures.

`bindFlags`

Type: [UINT](#)

A combination of [D3D11\\_BIND\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies how to bind the texture to pipeline stages. For a 1D texture, the allowable values are: [D3D11\\_BIND\\_SHADER\\_RESOURCE](#), [D3D11\\_BIND\\_RENDER\\_TARGET](#) and [D3D11\\_BIND\\_DEPTH\\_STENCIL](#).

`usage`

Type: [D3D11\\_USAGE](#)

A [D3D11\\_USAGE](#)-typed value that identifies how the texture is to be read from and written to.

`cpuAccessFlags`

Type: [UINT](#)

A combination of [D3D11\\_CPU\\_ACCESS\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies the types of CPU access allowed.

`miscFlags`

Type: [UINT](#)

A combination of [D3D11\\_RESOURCE\\_MISC\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value identifies other, less common resource options.

# Return value

None

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE1D\\_DESC](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# CD3D11\_TEXTURE1D\_DESC::~CD3D11\_TEXTURE1D\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_TEXTURE1D\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_TEXTURE1D_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE1D\\_DESC](#)

# CD3D11\_TEXTURE2D\_DESC class

Article 12/01/2017

Represents a 2D texture and provides convenience methods for creating 2D textures.

## Members

The CD3D11\_TEXTURE2D\_DESC class inherits from [D3D11\\_TEXTURE2D\\_DESC](#).

CD3D11\_TEXTURE2D\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_TEXTURE2D\_DESC class has these constructors.

[+] Expand table

Constructor	Description
<a href="#">CD3D11_TEXTURE2D_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_TEXTURE2D_DESC structure.
<a href="#">CD3D11_TEXTURE2D_DESC(D3D11_TEXTURE2D_DESC_values)</a> <a href="#">(DXGI_FORMAT,UINT,UINT,UINT,UINT,UINT,D3D11_USAGE,UINT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_TEXTURE2D_DESC structure that is initialized with <a href="#">D3D11_TEXTURE2D_DESC</a> values.

## Methods

The CD3D11\_TEXTURE2D\_DESC class has these methods.

[+] Expand table

Method	Description
<a href="#">~CD3D11_TEXTURE2D_DESC</a>	Destroys an instance of a CD3D11_TEXTURE2D_DESC structure.

Method	Description
<code>CD3D11_TEXTURE2D_DESC(D3D11_TEXTURE2D_DESC&amp;)</code> <code>(const &amp;D3D11_TEXTURE2D_DESC)</code>	Instantiates a new instance of a <code>CD3D11_TEXTURE2D_DESC</code> structure that is initialized with a <code>D3D11_TEXTURE2D_DESC</code> structure.
<code>D3D11_TEXTURE2D_DESC()</code>	This operator returns the address of a <code>D3D11_TEXTURE2D_DESC</code> structure that contains the data from the <code>CD3D11_TEXTURE2D_DESC</code> instance.

## Remarks

Here is how D3D11.h defines `CD3D11_TEXTURE2D_DESC`:

```
struct CD3D11_TEXTURE2D_DESC : public D3D11_TEXTURE2D_DESC
{
    CD3D11_TEXTURE2D_DESC()
    {}
    explicit CD3D11_TEXTURE2D_DESC( const D3D11_TEXTURE2D_DESC& o ) :
        D3D11_TEXTURE2D_DESC( o )
    {}
    explicit CD3D11_TEXTURE2D_DESC(
        DXGI_FORMAT format,
        UINT width,
        UINT height,
        UINT arraySize = 1,
        UINT mipLevels = 0,
        UINT bindFlags = D3D11_BIND_SHADER_RESOURCE,
        D3D11_USAGE usage = D3D11_USAGE_DEFAULT,
        UINT cpuaccessFlags = 0,
        UINT sampleCount = 1,
        UINT sampleQuality = 0,
        UINT miscFlags = 0 )
    {
        Width = width;
        Height = height;
        MipLevels = mipLevels;
        ArraySize = arraySize;
        Format = format;
        SampleDesc.Count = sampleCount;
        SampleDesc.Quality = sampleQuality;
        Usage = usage;
        BindFlags = bindFlags;
        CPUAccessFlags = cpuaccessFlags;
        MiscFlags = miscFlags;
    }
}
```

```
~CD3D11_TEXTURE2D_DESC() {}
operator const D3D11_TEXTURE2D_DESC&() const { return *this; }
};
```

# Requirements

[Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_TEXTURE2D\\_DESC](#) ↗

[CD3D11 Helper Structures](#) ↗

# CD3D11\_TEXTURE2D\_DESC::CD3D11\_TEXTURE2D\_DESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_TEXTURE2D\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_TEXTURE2D_DESC();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE2D\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_TEXTURE2D\_DESC::CD3D11\_TEXTURE2D\_DESC(const D3D11\_TEXTURE2D\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_TEXTURE2D\\_DESC](#) structure that is initialized with a [D3D11\\_TEXTURE2D\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_TEXTURE2D_DESC(
    [ref] const D3D11_TEXTURE2D_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_TEXTURE2D\\_DESC](#)

Address of the [D3D11\\_TEXTURE2D\\_DESC](#) structure that initializes the [CD3D11\\_TEXTURE2D\\_DESC](#) structure.

## Return value

None

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE2D\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_TEXTURE2D\_DESC::CD3D11\_TEXTURE2D\_DESC(DXGI\_FORMAT,UINT,UINT,UINT,UINT,UINT,D3D11\_USAGE,UINT,UINT,UINT) function (d3d11.h)

Article04/02/2021

Instantiates a new instance of a [CD3D11\\_TEXTURE2D\\_DESC](#) structure that is initialized with [D3D11\\_TEXTURE2D\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_TEXTURE2D_DESC(
    DXGI_FORMAT format,
    UINT width,
    UINT height,
    UINT arraySize,
    UINT mipLevels,
    UINT bindFlags,
    D3D11_USAGE usage,
    UINT cpuaccessFlags,
    UINT sampleCount,
    UINT sampleQuality,
    UINT miscFlags
);
```

## Parameters

`format`

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#)-typed value that specifies the texture format.

`width`

Type: [UINT](#)

Texture width (in texels). The range is from 1 to [D3D11\\_REQ\\_TEXTURE2D\\_U\\_OR\\_V\\_DIMENSION](#) (16384). For a texture cube-map, the

range is from 1 to D3D11\_REQ\_TEXTURECUBE\_DIMENSION (16384). However, the range is actually constrained by the [feature level](#) at which you create the rendering device.

`height`

Type: [UINT](#)

Texture height (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE2D\_U\_OR\_V\_DIMENSION (16384). For a texture cube-map, the range is from 1 to D3D11\_REQ\_TEXTURECUBE\_DIMENSION (16384). However, the range is actually constrained by the [feature level](#) at which you create the rendering device.

`arraySize`

Type: [UINT](#)

Number of textures in the texture array. The range is from 1 to D3D11\_REQ\_TEXTURE2D\_ARRAY\_AXIS\_DIMENSION (2048). For a texture cube-map, this value is a multiple of 6 (that is, 6 times the value in the **NumCubes** member of [D3D11\\_TEXCUBE\\_ARRAY\\_SRV](#)), and the range is from 6 to 2046. The range is actually constrained by the [feature level](#) at which you create the rendering device.

`mipLevels`

Type: [UINT](#)

The maximum number of mipmap levels in the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#). Use 1 for a multisampled texture; or 0 to generate a full set of subtextures.

`bindFlags`

Type: [UINT](#)

A combination of [D3D11\\_BIND\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies how to bind the texture to pipeline stages.

`usage`

Type: [D3D11\\_USAGE](#)

A [D3D11\\_USAGE](#)-typed value that identifies how the texture is to be read from and written to.

`cpuAccessFlags`

Type: **UINT**

A combination of [D3D11\\_CPU\\_ACCESS\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies the types of CPU access allowed.

`sampleCount`

Type: **UINT**

The sample count.

`sampleQuality`

Type: **UINT**

The sample quality.

`miscFlags`

Type: **UINT**

A combination of [D3D11\\_RESOURCE\\_MISC\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value identifies other, less common resource options. For a texture cube-map, set the [D3D11\\_RESOURCE\\_MISC\\_TEXTURECUBE](#) flag. Cube-map arrays (that is, *arraySize* > 6) require feature level [D3D\\_FEATURE\\_LEVEL\\_10\\_1](#) or higher.

## Return value

None

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE2D\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CD3D11\_TEXTURE2D\_DESC::~CD3D11\_TEXTURE2D\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_TEXTURE2D\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_TEXTURE2D_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE2D\\_DESC](#)

# CD3D11\_TEXTURE3D\_DESC class

Article 12/01/2017

Represents a 3D texture and provides convenience methods for creating 3D textures.

## Members

The CD3D11\_TEXTURE3D\_DESC class inherits from [D3D11\\_TEXTURE3D\\_DESC](#).

CD3D11\_TEXTURE3D\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_TEXTURE3D\_DESC class has these constructors.

[+] [Expand table](#)

Constructor	Description
<a href="#">CD3D11_TEXTURE3D_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_TEXTURE3D_DESC structure.
<a href="#">CD3D11_TEXTURE3D_DESC(D3D11_TEXTURE3D_DESC_values)</a> <a href="#">(DXGI_FORMAT,UINT,UINT,UINT,UINT,UINT,D3D11_USAGE,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_TEXTURE3D_DESC structure that is initialized with D3D11_TEXTURE3D_DESC values.

## Methods

The CD3D11\_TEXTURE3D\_DESC class has these methods.

[+] [Expand table](#)

Method	Description
<a href="#">~CD3D11_TEXTURE3D_DESC()</a>	Destroys an instance of a <b>CD3D11_TEXTURE3D_DESC</b> structure.
<b>CD3D11_TEXTURE3D_DESC(D3D11_TEXTURE3D_DESC&amp;)</b> <b>(const &amp;D3D11_TEXTURE3D_DESC)</b>	Instantiates a new instance of a <b>CD3D11_TEXTURE3D_DESC</b> structure that is initialized with a <b>D3D11_TEXTURE3D_DESC</b> structure.
<b>D3D11_TEXTURE3D_DESC()</b>	This operator returns the address of a <b>D3D11_TEXTURE3D_DESC</b> structure that contains the data from the <b>CD3D11_TEXTURE3D_DESC</b> instance.

## Remarks

Here is how D3D11.h defines **CD3D11\_TEXTURE3D\_DESC**:

```
struct CD3D11_TEXTURE3D_DESC : public D3D11_TEXTURE3D_DESC
{
    CD3D11_TEXTURE3D_DESC()
    {}
    explicit CD3D11_TEXTURE3D_DESC( const D3D11_TEXTURE3D_DESC& o ) :
        D3D11_TEXTURE3D_DESC( o )
    {}
    explicit CD3D11_TEXTURE3D_DESC(
        DXGI_FORMAT format,
        UINT width,
        UINT height,
        UINT depth,
        UINT mipLevels = 0,
        UINT bindFlags = D3D11_BIND_SHADER_RESOURCE,
        D3D11_USAGE usage = D3D11_USAGE_DEFAULT,
        UINT cpuaccessFlags = 0,
        UINT miscFlags = 0 )
    {
        Width = width;
        Height = height;
        Depth = depth;
        MipLevels = mipLevels;
        Format = format;
        Usage = usage;
        BindFlags = bindFlags;
        CPUAccessFlags = cpuaccessFlags;
        MiscFlags = miscFlags;
    }
}
```

```
    }
~CD3D11_TEXTURE3D_DESC() {}
operator const D3D11_TEXTURE3D_DESC&() const { return *this; }
};
```

# Requirements

 Expand table

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_TEXTURE3D\\_DESC](#) ↗

[CD3D11 Helper Structures](#) ↗

# CD3D11\_TEXTURE3D\_DESC::CD3D11\_TEXTURE3D\_DESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_TEXTURE3D\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_TEXTURE3D_DESC();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE3D\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_TEXTURE3D\_DESC::CD3D11\_TEXTURE3D\_DESC(const D3D11\_TEXTURE3D\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_TEXTURE3D\\_DESC](#) structure that is initialized with a [D3D11\\_TEXTURE3D\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_TEXTURE3D_DESC(
    [ref] const D3D11_TEXTURE3D_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_TEXTURE3D\\_DESC](#)

Address of the [D3D11\\_TEXTURE3D\\_DESC](#) structure that initializes the [CD3D11\\_TEXTURE3D\\_DESC](#) structure.

## Return value

None

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE3D\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_TEXTURE3D\_DESC::CD3D11\_TEXTURE3D\_DESC(DXGI\_FORMAT,UINT,UINT,UINT,UINT,UINT,D3D11\_USAGE,UINT,UINT) function (d3d11.h)

Article04/02/2021

Instantiates a new instance of a [CD3D11\\_TEXTURE3D\\_DESC](#) structure that is initialized with [D3D11\\_TEXTURE3D\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_TEXTURE3D_DESC(
    DXGI_FORMAT format,
    UINT width,
    UINT height,
    UINT depth,
    UINT mipLevels,
    UINT bindFlags,
    D3D11_USAGE usage,
    UINT cpuAccessFlags,
    UINT miscFlags
);
```

## Parameters

`format`

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#)-typed value that specifies the texture format.

`width`

Type: [UINT](#)

Texture width (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE3D\_U\_V\_OR\_W\_DIMENSION (2048). However, the range is actually constrained by the [feature level](#) at which you create the rendering device.

`height`

Type: [UINT](#)

Texture height (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE3D\_U\_V\_OR\_W\_DIMENSION (2048). However, the range is actually constrained by the [feature level](#) at which you create the rendering device.

`depth`

Type: [UINT](#)

Texture depth (in texels). The range is from 1 to D3D11\_REQ\_TEXTURE3D\_U\_V\_OR\_W\_DIMENSION (2048). However, the range is actually constrained by the [feature level](#) at which you create the rendering device.

`mipLevels`

Type: [UINT](#)

The maximum number of mipmap levels in the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#). Use 1 for a multisampled texture; or 0 to generate a full set of subtextures.

`bindFlags`

Type: [UINT](#)

A combination of [D3D11\\_BIND\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies how to bind the texture to pipeline stages.

`usage`

Type: [D3D11\\_USAGE](#)

A [D3D11\\_USAGE](#)-typed value that identifies how the texture is to be read from and written to.

`cpuAccessFlags`

Type: [UINT](#)

A combination of [D3D11\\_CPU\\_ACCESS\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies the types of CPU access allowed.

`miscFlags`

Type: [UINT](#)

A combination of [D3D11\\_RESOURCE\\_MISC\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value identifies other, less common resource options.

## Return value

None

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE3D\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CD3D11\_TEXTURE3D\_DESC::~CD3D11\_TEXTURE3D\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_TEXTURE3D\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_TEXTURE3D_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_TEXTURE3D\\_DESC](#)

# CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC class

Article 12/01/2017

Represents a shader-resource view and provides convenience methods for creating shader-resource views.

## Members

The CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC class inherits from [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#).

CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC class has these constructors.

 Expand table

Constructor	Description
<a href="#">CD3D11_SHADER_RESOURCE_VIEW_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_SHADER_RESOURCE_VIEW_DESC structure.
<a href="#">CD3D11_SHADER_RESOURCE_VIEW_DESC(D3D11_SHADER_RESOURCE_VIEW_DESC_values1)(D3D11_SRV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_SHADER_RESOURCE_VIEW_DESC structure that is initialized with <a href="#">D3D11_SHADER_RESOURCE_VIEW_DESC</a> values.
<a href="#">CD3D11_SHADER_RESOURCE_VIEW_DESC(D3D11_SHADER_RESOURCE_VIEW_DESC_values2)(ID3D11Buffer,DXGI_FORMAT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_SHADER_RESOURCE_VIEW_DESC structure that is initialized with <a href="#">D3D11_BUFFEREX_SRV</a> values.
<a href="#">CD3D11_SHADER_RESOURCE_VIEW_DESC(D3D11_SHADER_RESOURCE_VIEW_DESC_values3)(ID3D11Texture1D,D3D11_SRV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_SHADER_RESOURCE_VIEW_DESC structure that is initialized with <a href="#">D3D11_TEX1D_SRV</a> or <a href="#">D3D11_TEX1D_ARRAY_SRV</a> values.
<a href="#">CD3D11_SHADER_RESOURCE_VIEW_DESC(D3D11_SHADER_RESOURCE_VIEW_DESC_values4)(ID3D11Texture2D,D3D11_SRV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_SHADER_RESOURCE_VIEW_DESC structure that is initialized with 2D texture values.
<a href="#">CD3D11_SHADER_RESOURCE_VIEW_DESC(D3D11_SHADER_RESOURCE_VIEW_DESC_values5)(ID3D11Texture3D,DXGI_FORMAT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_SHADER_RESOURCE_VIEW_DESC structure that is initialized with 3D texture values.

## Methods

The CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC class has these methods.

 Expand table

Method	Description
<a href="#">~CD3D11_SHADER_RESOURCE_VIEW_DESC()</a>	Destroys an instance of a CD3D11_SHADER_RESOURCE_VIEW_DESC structure.
<a href="#">CD3D11_SHADER_RESOURCE_VIEW_DESC(D3D11_SHADER_RESOURCE_VIEW_DESC)</a> <a href="#">(const &amp;D3D11_SHADER_RESOURCE_VIEW_DESC)</a>	Instantiates a new instance of a CD3D11_SHADER_RESOURCE_VIEW_DESC structure that is initialized with a D3D11_SHADER_RESOURCE_VIEW_DESC structure.
<a href="#">D3D11_SHADER_RESOURCE_VIEW_DESC()</a>	This operator returns the address of a D3D11_SHADER_RESOURCE_VIEW_DESC structure that contains the data from the CD3D11_SHADER_RESOURCE_VIEW_DESC instance.

## Remarks

Here is how D3D11.h defines CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC:

```
struct CD3D11_SHADER_RESOURCE_VIEW_DESC : public D3D11_SHADER_RESOURCE_VIEW_DESC
{
    CD3D11_SHADER_RESOURCE_VIEW_DESC()
    {}

    explicit CD3D11_SHADER_RESOURCE_VIEW_DESC( const D3D11_SHADER_RESOURCE_VIEW_DESC& o ) :
        D3D11_SHADER_RESOURCE_VIEW_DESC( o )
    {}

    explicit CD3D11_SHADER_RESOURCE_VIEW_DESC(
        D3D11_SRV_DIMENSION viewDimension,
        DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
        UINT mostDetailedMip = 0, // FirstElement for BUFFER
        UINT mipLevels = -1, // NumElements for BUFFER
        UINT firstArraySlice = 0, // First2DArrayFace for TEXTURECUBEARRAY
        UINT arraySize = -1, // NumCubes for TEXTURECUBEARRAY
        UINT flags = 0 ) // BUFFEREX only
    {
        Format = format;
        ViewDimension = viewDimension;
        switch (viewDimension)
        {
            case D3D11_SRV_DIMENSION_BUFFER:
                Buffer.FirstElement = mostDetailedMip;
                Buffer.NumElements = mipLevels;
                break;
            case D3D11_SRV_DIMENSION_TEXTURE1D:
                Texture1D.MostDetailedMip = mostDetailedMip;
                Texture1D.MipLevels = mipLevels;
                break;
            case D3D11_SRV_DIMENSION_TEXTURE1DARRAY:
                Texture1DArray.MostDetailedMip = mostDetailedMip;
                Texture1DArray.MipLevels = mipLevels;
                Texture1DArray.FirstArraySlice = firstArraySlice;
                break;
        }
    }
}
```

```

        Texture1DArray.ArraySize = arraySize;
        break;
    case D3D11_SRV_DIMENSION_TEXTURE2D:
        Texture2D.MostDetailedMip = mostDetailedMip;
        Texture2D.MipLevels = mipLevels;
        break;
    case D3D11_SRV_DIMENSION_TEXTURE2DARRAY:
        Texture2DArray.MostDetailedMip = mostDetailedMip;
        Texture2DArray.MipLevels = mipLevels;
        Texture2DArray.FirstArraySlice = firstArraySlice;
        Texture2DArray.ArraySize = arraySize;
        break;
    case D3D11_SRV_DIMENSION_TEXTURE2DMS:
        break;
    case D3D11_SRV_DIMENSION_TEXTURE2DMSARRAY:
        Texture2DMSArray.FirstArraySlice = firstArraySlice;
        Texture2DMSArray.ArraySize = arraySize;
        break;
    case D3D11_SRV_DIMENSION_TEXTURE3D:
        Texture3D.MostDetailedMip = mostDetailedMip;
        Texture3D.MipLevels = mipLevels;
        break;
    case D3D11_SRV_DIMENSION_TEXTURECUBE:
        TextureCube.MostDetailedMip = mostDetailedMip;
        TextureCube.MipLevels = mipLevels;
        break;
    case D3D11_SRV_DIMENSION_TEXTURECUBEARRAY:
        TextureCubeArray.MostDetailedMip = mostDetailedMip;
        TextureCubeArray.MipLevels = mipLevels;
        TextureCubeArray.First2DArrayFace = firstArraySlice;
        TextureCubeArray.NumCubes = arraySize;
        break;
    case D3D11_SRV_DIMENSION_BUFFEREX:
        BufferEx.FirstElement = mostDetailedMip;
        BufferEx.NumElements = mipLevels;
        BufferEx.Flags = flags;
        break;
    default: break;
}
}

explicit CD3D11_SHADER_RESOURCE_VIEW_DESC(
    _In_ ID3D11Buffer*,
    DXGI_FORMAT format,
    UINT firstElement,
    UINT numElements,
    UINT flags = 0 )
{
    Format = format;
    ViewDimension = D3D11_SRV_DIMENSION_BUFFEREX;
    BufferEx.FirstElement = firstElement;
    BufferEx.NumElements = numElements;
    BufferEx.Flags = flags;
}

explicit CD3D11_SHADER_RESOURCE_VIEW_DESC(
    _In_ ID3D11Texture1D* pTex1D,
    D3D11_SRV_DIMENSION viewDimension,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mostDetailedMip = 0,
    UINT mipLevels = -1,
    UINT firstArraySlice = 0,
    UINT arraySize = -1 )
{
    ViewDimension = viewDimension;
    if (DXGI_FORMAT_UNKNOWN == format || -1 == mipLevels ||
        (-1 == arraySize && D3D11_SRV_DIMENSION_TEXTURE1DARRAY == viewDimension))
    {
        D3D11_TEXTURE1D_DESC TexDesc;
        pTex1D->GetDesc( &TexDesc );

```

```

        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == mipLevels) mipLevels = TexDesc.MipLevels - mostDetailedMip;
        if (-1 == arraySize) arraySize = TexDesc.ArraySize - firstArraySlice;
    }
    Format = format;
    switch (viewDimension)
    {
    case D3D11_SRV_DIMENSION_TEXTURE1D:
        Texture1D.MostDetailedMip = mostDetailedMip;
        Texture1D.MipLevels = mipLevels;
        break;
    case D3D11_SRV_DIMENSION_TEXTURE1DARRAY:
        Texture1DArray.MostDetailedMip = mostDetailedMip;
        Texture1DArray.MipLevels = mipLevels;
        Texture1DArray.FirstArraySlice = firstArraySlice;
        Texture1DArray.ArraySize = arraySize;
        break;
    default: break;
    }
}
explicit CD3D11_SHADER_RESOURCE_VIEW_DESC(
    _In_ ID3D11Texture2D* pTex2D,
    D3D11_SRV_DIMENSION viewDimension,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mostDetailedMip = 0,
    UINT mipLevels = -1,
    UINT firstArraySlice = 0, // First2DArrayFace for TEXTURECUBEARRAY
    UINT arraySize = -1 ) // NumCubes for TEXTURECUBEARRAY
{
    ViewDimension = viewDimension;
    if (DXGI_FORMAT_UNKNOWN == format ||
        (-1 == mipLevels &&
            D3D11_SRV_DIMENSION_TEXTURE2DMS != viewDimension &&
            D3D11_SRV_DIMENSION_TEXTURE2DMSARRAY != viewDimension) ||
        (-1 == arraySize &&
            (D3D11_SRV_DIMENSION_TEXTURE2DARRAY == viewDimension ||
            D3D11_SRV_DIMENSION_TEXTURE2DMSARRAY == viewDimension ||
            D3D11_SRV_DIMENSION_TEXTURECUBEARRAY == viewDimension)))
    {
        D3D11_TEXTURE2D_DESC TexDesc;
        pTex2D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == mipLevels) mipLevels = TexDesc.MipLevels - mostDetailedMip;
        if (-1 == arraySize)
        {
            arraySize = TexDesc.ArraySize - firstArraySlice;
            if (D3D11_SRV_DIMENSION_TEXTURECUBEARRAY == viewDimension) arraySize /= 6;
        }
    }
    Format = format;
    switch (viewDimension)
    {
    case D3D11_SRV_DIMENSION_TEXTURE2D:
        Texture2D.MostDetailedMip = mostDetailedMip;
        Texture2D.MipLevels = mipLevels;
        break;
    case D3D11_SRV_DIMENSION_TEXTURE2DARRAY:
        Texture2DArray.MostDetailedMip = mostDetailedMip;
        Texture2DArray.MipLevels = mipLevels;
        Texture2DArray.FirstArraySlice = firstArraySlice;
        Texture2DArray.ArraySize = arraySize;
        break;
    case D3D11_SRV_DIMENSION_TEXTURE2DMS:
        break;
    case D3D11_SRV_DIMENSION_TEXTURE2DMSARRAY:
        Texture2DMSArray.FirstArraySlice = firstArraySlice;
        Texture2DMSArray.ArraySize = arraySize;
        break;
    }
}

```

```

        case D3D11_SRV_DIMENSION_TEXTURECUBE:
            TextureCube.MostDetailedMip = mostDetailedMip;
            TextureCube.MipLevels = mipLevels;
            break;
        case D3D11_SRV_DIMENSION_TEXTURECUBEARRAY:
            TextureCubeArray.MostDetailedMip = mostDetailedMip;
            TextureCubeArray.MipLevels = mipLevels;
            TextureCubeArray.First2DArrayFace = firstArraySlice;
            TextureCubeArray.NumCubes = arraySize;
            break;
        default: break;
    }
}

explicit CD3D11_SHADER_RESOURCE_VIEW_DESC(
    _In_ ID3D11Texture3D* pTex3D,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mostDetailedMip = 0,
    UINT mipLevels = -1 )
{
    ViewDimension = D3D11_SRV_DIMENSION_TEXTURE3D;
    if (DXGI_FORMAT_UNKNOWN == format || -1 == mipLevels)
    {
        D3D11_TEXTURE3D_DESC TexDesc;
        pTex3D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == mipLevels) mipLevels = TexDesc.MipLevels - mostDetailedMip;
    }
    Format = format;
    Texture3D.MostDetailedMip = mostDetailedMip;
    Texture3D.MipLevels = mipLevels;
}
~CD3D11_SHADER_RESOURCE_VIEW_DESC() {}
operator const D3D11_SHADER_RESOURCE_VIEW_DESC&() const { return *this; }
};


```

## Requirements

[ ] [Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) ↗

[CD3D11 Helper Structures](#) ↗

# CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC function (d3d11.h)

Article02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC();
```

## Return value

None

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC::CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC( const D3D11\_SHADER\_RESOURCE\_VIEW\_DESC& ) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure that is initialized with a [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC(
    [ref] const D3D11_SHADER_RESOURCE_VIEW_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

Address of the [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure that initializes the [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure.

## Return value

None

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC

## C::CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC(D3D11\_SRV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT,UINT,UINT)

### function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure that is initialized with [D3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC(
    D3D11_SRV_DIMENSION viewDimension,
    DXGI_FORMAT         format,
    UINT                mostDetailedMip,
    UINT                mipLevels,
    UINT                firstArraySlice,
    UINT                arraySize,
    UINT                flags
);
```

## Parameters

`viewDimension`

Type: [D3D11\\_SRV\\_DIMENSION](#)

A [D3D11\\_SRV\\_DIMENSION](#)-typed value that specifies the resource type of the view.

`format`

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#)-typed value that specifies the viewing format.

`mostDetailedMip`

Type: [UINT](#)

Index of the most detailed mipmap level to use; this number is from 0 to *mipLevels* - 1. For a buffer, this is the number of bytes between the beginning of the buffer and the first element to access.

FirstElement of [D3D11\\_BUFFER\\_SRV](#).

`mipLevels`

Type: [UINT](#)

The maximum number of mipmap levels for the view. For a buffer, this is the total number of elements in the view. **NumElements** of [D3D11\\_BUFFER\\_SRV](#).

`firstArraySlice`

Type: [UINT](#)

The index of the first element to use in an array of elements.

First2DArrayFace of [D3D11\\_TEXCUBE\\_ARRAY\\_SRV](#).

`arraySize`

Type: [UINT](#)

Number of elements in the array. **NumCubes** of [D3D11\\_TEXCUBE\\_ARRAY\\_SRV](#).

`flags`

Type: [UINT](#)

A [D3D11\\_BUFFEREX\\_SRV\\_FLAG](#)-typed value that identifies view options for a buffer. For [D3D11\\_BUFFEREX\\_SRV](#) only.

## Return value

None

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC::CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC(ID3D11Buffer\*,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure that is initialized with [D3D11\\_BUFFEREX\\_SRV](#) values.

## Syntax

C++

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC(
    ID3D11Buffer *unnamedParam1,
    DXGI_FORMAT    format,
    UINT           firstElement,
    UINT           numElements,
    UINT           flags
);
```

## Parameters

unnamedParam1

A pointer to a [ID3D11Buffer](#) interface for a buffer.

format

A [DXGI\\_FORMAT](#)-typed value that specifies the viewing format.

firstElement

Number of bytes between the beginning of the buffer and the first element to access.

numElements

Total number of elements in the view.

flags

A D3D11\_BUFFEREX\_SRV\_FLAG-typed value that identifies view options for a buffer.

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# **CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC** C::CD3D11\_SHADER\_RESOURCE\_VIEW\_D ESC(ID3D11Texture1D\*,D3D11\_SRV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure that is initialized with **D3D11\_TEX1D\_SRV** or **D3D11\_TEX1D\_ARRAY\_SRV** values.

## Syntax

C++

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC(
    ID3D11Texture1D      *pTex1D,
    D3D11_SRV_DIMENSION  viewDimension,
    DXGI_FORMAT           format,
    UINT                  mostDetailedMip,
    UINT                  mipLevels,
    UINT                  firstArraySlice,
    UINT                  arraySize
);
```

## Parameters

`pTex1D`

A pointer to a **ID3D11Texture1D** interface for a 1D texture.

`viewDimension`

A **D3D11\_SRV\_DIMENSION**-typed value that specifies the resource type of the view.

`format`

A **DXGI\_FORMAT**-typed value that specifies the viewing format.

`mostDetailedMip`

Index of the most detailed mipmap level to use; this number is from 0 to *mipLevels* -1.

`mipLevels`

The maximum number of mipmap levels for the view.

`firstArraySlice`

The index of the first element to use in an array of elements.

`arraySize`

Number of elements in the array.

## Return value

None

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC::CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC(ID3D11Texture1D\*,D3D11\_SRV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure that is initialized with **D3D11\_TEX1D\_SRV** or **D3D11\_TEX1D\_ARRAY\_SRV** values.

## Syntax

C++

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC(
    ID3D11Texture1D      *pTex1D,
    D3D11_SRV_DIMENSION  viewDimension,
    DXGI_FORMAT           format,
    UINT                  mostDetailedMip,
    UINT                  mipLevels,
    UINT                  firstArraySlice,
    UINT                  arraySize
);
```

## Parameters

`pTex1D`

A pointer to a **ID3D11Texture1D** interface for a 1D texture.

`viewDimension`

A **D3D11\_SRV\_DIMENSION**-typed value that specifies the resource type of the view.

`format`

A **DXGI\_FORMAT**-typed value that specifies the viewing format.

`mostDetailedMip`

Index of the most detailed mipmap level to use; this number is from 0 to *mipLevels* -1.

`mipLevels`

The maximum number of mipmap levels for the view.

`firstArraySlice`

The index of the first element to use in an array of elements.

`arraySize`

Number of elements in the array.

## Return value

None

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# **CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC** C::CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC(ID3D11Texture3D\*,DXGI\_FORMAT,UINT,UINT) function (d3d11.h)

Article 08/31/2022

Instantiates a new instance of a [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure that is initialized with 3D texture values.

## Syntax

C++

```
void CD3D11_SHADER_RESOURCE_VIEW_DESC(
    ID3D11Texture3D *pTex3D,
    DXGI_FORMAT      format,
    UINT             mostDetailedMip,
    UINT             mipLevels
);
```

## Parameters

`pTex3D`

A pointer to a `ID3D11Texture3D` interface for a 3D texture.

`format`

A `DXGI_FORMAT`-typed value that specifies the viewing format.

`mostDetailedMip`

Index of the most detailed mipmap level to use; this number is from 0 to *mipLevels* -1.

`mipLevels`

The maximum number of mipmap levels for the view.

## Return value

None

# Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC::~CD3D11\_SHADER\_RESOURCE\_VIEW\_DESC function (d3d11.h)

Article 02/22/2024

Destroys an instance of a [CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_SHADER_RESOURCE_VIEW_DESC();
```

## Return value

None

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SHADER\\_RESOURCE\\_VIEW\\_DESC](#)

# CD3D11\_RENDER\_TARGET\_VIEW\_DESC class

Article 12/01/2017

Represents a render-target view and provides convenience methods for creating render-target views.

## Members

The CD3D11\_RENDER\_TARGET\_VIEW\_DESC class inherits from [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#).

CD3D11\_RENDER\_TARGET\_VIEW\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_RENDER\_TARGET\_VIEW\_DESC class has these constructors.

[Expand table](#)

Constructor	Description
<a href="#">CD3D11_RENDER_TARGET_VIEW_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_RENDER_TARGET_VIEW_DESC structure.
<a href="#">CD3D11_RENDER_TARGET_VIEW_DESC(D3D11_RENDER_TARGET_VIEW_DESC_values)</a> <a href="#">(D3D11_RTV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_RENDER_TARGET_VIEW_DESC structure that is initialized with D3D11_RENDER_TARGET_VIEW_DESC values.
<a href="#">CD3D11_RENDER_TARGET_VIEW_DESC(D3D11_RENDER_TARGET_VIEW_DESC_values2)</a> <a href="#">(ID3D11Buffer,DXGI_FORMAT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_RENDER_TARGET_VIEW_DESC structure that is initialized with D3D11_BUFFER_RTV values.
<a href="#">CD3D11_RENDER_TARGET_VIEW_DESC(D3D11_RENDER_TARGET_VIEW_DESC_values3)</a> <a href="#">(ID3D11Texture1D,D3D11_RTV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_RENDER_TARGET_VIEW_DESC structure that is initialized with D3D11_TEX1D_RTV or D3D11_TEX1D_ARRAY_RTV values.
<a href="#">CD3D11_RENDER_TARGET_VIEW_DESC(D3D11_RENDER_TARGET_VIEW_DESC_values4)</a> <a href="#">(ID3D11Texture2D,D3D11_RTV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_RENDER_TARGET_VIEW_DESC structure that is initialized with 2D texture values.
<a href="#">CD3D11_RENDER_TARGET_VIEW_DESC(D3D11_RENDER_TARGET_VIEW_DESC_values5)</a> <a href="#">(ID3D11Texture3D,DXGI_FORMAT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_RENDER_TARGET_VIEW_DESC structure that is initialized with 3D texture values.

## Methods

The `CD3D11_RENDER_TARGET_VIEW_DESC` class has these methods.

[+] Expand table

Method	Description
<code>~CD3D11_RENDER_TARGET_VIEW_DESC()</code>	Destroys an instance of a <code>CD3D11_RENDER_TARGET_VIEW_DESC</code> structure.
<code>CD3D11_RENDER_TARGET_VIEW_DESC(D3D11_RENDER_TARGET_VIEW_DESC&amp;)</code> <code>(const &amp;D3D11_RENDER_TARGET_VIEW_DESC)</code>	Instantiates a new instance of a <code>CD3D11_RENDER_TARGET_VIEW_DESC</code> structure that is initialized with a <code>D3D11_RENDER_TARGET_VIEW_DESC</code> structure.
<code>D3D11_RENDER_TARGET_VIEW_DESC()</code>	This operator returns the address of a <code>D3D11_RENDER_TARGET_VIEW_DESC</code> structure that contains the data from the <code>CD3D11_RENDER_TARGET_VIEW_DESC</code> instance.

## Remarks

Here is how D3D11.h defines `CD3D11_RENDER_TARGET_VIEW_DESC`:

```
struct CD3D11_RENDER_TARGET_VIEW_DESC : public D3D11_RENDER_TARGET_VIEW_DESC
{
    CD3D11_RENDER_TARGET_VIEW_DESC()
    {}
    explicit CD3D11_RENDER_TARGET_VIEW_DESC( const D3D11_RENDER_TARGET_VIEW_DESC& o ) :
        D3D11_RENDER_TARGET_VIEW_DESC( o )
    {}
    explicit CD3D11_RENDER_TARGET_VIEW_DESC(
        D3D11_RTV_DIMENSION viewDimension,
        DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
        UINT mipSlice = 0, // FirstElement for BUFFER
        UINT firstArraySlice = 0, // NumElements for BUFFER, FirstWSlice for TEXTURE3D
        UINT arraySize = -1 ) // WSize for TEXTURE3D
    {
        Format = format;
        ViewDimension = viewDimension;
        switch (viewDimension)
        {
            case D3D11_RTV_DIMENSION_BUFFER:
                Buffer.FirstElement = mipSlice;
                Buffer.NumElements = firstArraySlice;
                break;
            case D3D11_RTV_DIMENSION_TEXTURE1D:
                Texture1D.MipSlice = mipSlice;
                break;
        }
    }
}
```

```

        case D3D11_RT_V_DIMENSION_TEXTURE1DARRAY:
            Texture1DArray.MipSlice = mipSlice;
            Texture1DArray.FirstArraySlice = firstArraySlice;
            Texture1DArray.ArraySize = arraySize;
            break;
        case D3D11_RT_V_DIMENSION_TEXTURE2D:
            Texture2D.MipSlice = mipSlice;
            break;
        case D3D11_RT_V_DIMENSION_TEXTURE2DARRAY:
            Texture2DArray.MipSlice = mipSlice;
            Texture2DArray.FirstArraySlice = firstArraySlice;
            Texture2DArray.ArraySize = arraySize;
            break;
        case D3D11_RT_V_DIMENSION_TEXTURE2DMS:
            break;
        case D3D11_RT_V_DIMENSION_TEXTURE2DMSARRAY:
            Texture2DMSArray.FirstArraySlice = firstArraySlice;
            Texture2DMSArray.ArraySize = arraySize;
            break;
        case D3D11_RT_V_DIMENSION_TEXTURE3D:
            Texture3D.MipSlice = mipSlice;
            Texture3D.FirstWSlice = firstArraySlice;
            Texture3D.WSize = arraySize;
            break;
        default: break;
    }
}

explicit CD3D11_RENDER_TARGET_VIEW_DESC(
    _In_ ID3D11Buffer*,
    DXGI_FORMAT format,
    UINT firstElement,
    UINT numElements )
{
    Format = format;
    ViewDimension = D3D11_RT_V_DIMENSION_BUFFER;
    Buffer.FirstElement = firstElement;
    Buffer.NumElements = numElements;
}

explicit CD3D11_RENDER_TARGET_VIEW_DESC(
    _In_ ID3D11Texture1D* pTex1D,
    D3D11_RT_V_DIMENSION viewDimension,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mipSlice = 0,
    UINT firstArraySlice = 0,
    UINT arraySize = -1 )
{
    ViewDimension = viewDimension;
    if (DXGI_FORMAT_UNKNOWN == format ||
        (-1 == arraySize && D3D11_RT_V_DIMENSION_TEXTURE1DARRAY == viewDimension))
    {
        D3D11_TEXTURE1D_DESC TexDesc;
        pTex1D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == arraySize) arraySize = TexDesc.ArraySize - firstArraySlice;
    }
    Format = format;
    switch (viewDimension)
    {
        case D3D11_RT_V_DIMENSION_TEXTURE1D:
            Texture1D.MipSlice = mipSlice;
            break;
        case D3D11_RT_V_DIMENSION_TEXTURE1DARRAY:
            Texture1DArray.MipSlice = mipSlice;
            Texture1DArray.FirstArraySlice = firstArraySlice;

```

```

        Texture1DArray.ArraySize = arraySize;
        break;
    default: break;
}
}

explicit CD3D11_RENDER_TARGET_VIEW_DESC(
    _In_ ID3D11Texture2D* pTex2D,
    D3D11_RTV_DIMENSION viewDimension,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mipSlice = 0,
    UINT firstArraySlice = 0,
    UINT arraySize = -1 )
{
    ViewDimension = viewDimension;
    if (DXGI_FORMAT_UNKNOWN == format ||
        (-1 == arraySize &&
            (D3D11_RTV_DIMENSION_TEXTURE2DARRAY == viewDimension ||
            D3D11_RTV_DIMENSION_TEXTURE2DMSARRAY == viewDimension)))
    {
        D3D11_TEXTURE2D_DESC TexDesc;
        pTex2D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == arraySize) arraySize = TexDesc.ArraySize - firstArraySlice;
    }
    Format = format;
    switch (viewDimension)
    {
        case D3D11_RTV_DIMENSION_TEXTURE2D:
            Texture2D.MipSlice = mipSlice;
            break;
        case D3D11_RTV_DIMENSION_TEXTURE2DARRAY:
            Texture2DArray.MipSlice = mipSlice;
            Texture2DArray.FirstArraySlice = firstArraySlice;
            Texture2DArray.ArraySize = arraySize;
            break;
        case D3D11_RTV_DIMENSION_TEXTURE2DMS:
            break;
        case D3D11_RTV_DIMENSION_TEXTURE2DMSARRAY:
            Texture2DMSArray.FirstArraySlice = firstArraySlice;
            Texture2DMSArray.ArraySize = arraySize;
            break;
        default: break;
    }
}

explicit CD3D11_RENDER_TARGET_VIEW_DESC(
    _In_ ID3D11Texture3D* pTex3D,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mipSlice = 0,
    UINT firstWSlice = 0,
    UINT wSize = -1 )
{
    ViewDimension = D3D11_RTV_DIMENSION_TEXTURE3D;
    if (DXGI_FORMAT_UNKNOWN == format || -1 == wSize)
    {
        D3D11_TEXTURE3D_DESC TexDesc;
        pTex3D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == wSize) wSize = TexDesc.Depth - firstWSlice;
    }
    Format = format;
    Texture3D.MipSlice = mipSlice;
    Texture3D.FirstWSlice = firstWSlice;
    Texture3D.WSize = wSize;
}

```

```
~CD3D11_RENDER_TARGET_VIEW_DESC() {}
operator const D3D11_RENDER_TARGET_VIEW_DESC&() const { return *this; }
};
```

# Requirements

 [Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

[CD3D11 Helper Structures](#)

# CD3D11\_RENDER\_TARGET\_VIEW\_DESC::CD3D11\_RENDER\_TARGET\_VIEW\_DESC function (d3d11.h)

Article02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_RENDER_TARGET_VIEW_DESC();
```

## Return value

None

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_RENDER\_TARGET\_VIEW\_DESC::CD3D11\_RENDER\_TARGET\_VIEW\_DESC(const CD3D11\_RENDER\_TARGET\_VIEW\_DESC &) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure that is initialized with a [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_RENDER_TARGET_VIEW_DESC(
    [ref] const D3D11_RENDER_TARGET_VIEW_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

Address of the [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure that initializes the [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure.

## Return value

None

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_RENDER\_TARGET\_VIEW\_DESC::CD3D11\_RENDER\_TARGET\_VIEW\_DESC(D3D11\_RTVDIMENSION,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure that is initialized with [D3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_RENDER_TARGET_VIEW_DESC(
    D3D11_RTVDIMENSION viewDimension,
    DXGI_FORMAT         format,
    UINT                mipSlice,
    UINT                firstArraySlice,
    UINT                arraySize
);
```

## Parameters

`viewDimension`

Type: [D3D11\\_RTVDIMENSION](#)

A [D3D11\\_RTVDIMENSION](#)-typed value that specifies the resource type of the view.

`format`

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#)-typed value that specifies the viewing format.

`mipSlice`

Type: [UINT](#)

The index of the mipmap level to use mip slice. **FirstElement** for [D3D11\\_BUFFER\\_SRV](#).

`firstArraySlice`

Type: [UINT](#)

The index of the first element to use in an array of elements.

NumElements for [D3D11\\_BUFFER\\_SRV](#). FirstWSlice for [D3D11\\_TEX3D\\_RTV](#).

`arraySize`

Type: [UINT](#)

Number of elements in the array. WSize for [D3D11\\_TEX3D\\_RTV](#).

## Return value

None

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# **CD3D11\_RENDER\_TARGET\_VIEW\_DESC::CD3D11\_RENDER\_TARGET\_VIEW\_DESC(ID3D11Texture1D\*,D3D11\_RTV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT)**

## **function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure that is initialized with **D3D11\_TEX1D\_RTV** or **D3D11\_TEX1D\_ARRAY\_RTV** values.

## Syntax

C++

```
void CD3D11_RENDER_TARGET_VIEW_DESC(
    ID3D11Texture1D      *pTex1D,
    D3D11_RTV_DIMENSION viewDimension,
    DXGI_FORMAT           format,
    UINT                  mipSlice,
    UINT                  firstArraySlice,
    UINT                  arraySize
);
```

## Parameters

**pTex1D**

A pointer to a **ID3D11Texture1D** interface for a 1D texture.

**viewDimension**

A **D3D11\_RTV\_DIMENSION**-typed value that specifies the resource type of the view.

**format**

A **DXGI\_FORMAT**-typed value that specifies the viewing format.

**mipSlice**

The index of the mipmap level to use mip slice.

`firstArraySlice`

The index of the first element to use in an array of elements.

`arraySize`

Number of elements in the array.

## Return value

None

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

# **CD3D11\_RENDER\_TARGET\_VIEW\_DESC::CD3D11\_RENDER\_TARGET\_VIEW\_DESC(ID3D11Texture1D\*,D3D11\_RTV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT)**

## **function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure that is initialized with **D3D11\_TEX1D\_RTV** or **D3D11\_TEX1D\_ARRAY\_RTV** values.

## Syntax

C++

```
void CD3D11_RENDER_TARGET_VIEW_DESC(
    ID3D11Texture1D      *pTex1D,
    D3D11_RTV_DIMENSION viewDimension,
    DXGI_FORMAT           format,
    UINT                  mipSlice,
    UINT                  firstArraySlice,
    UINT                  arraySize
);
```

## Parameters

**pTex1D**

A pointer to a **ID3D11Texture1D** interface for a 1D texture.

**viewDimension**

A **D3D11\_RTV\_DIMENSION**-typed value that specifies the resource type of the view.

**format**

A **DXGI\_FORMAT**-typed value that specifies the viewing format.

**mipSlice**

The index of the mipmap level to use mip slice.

`firstArraySlice`

The index of the first element to use in an array of elements.

`arraySize`

Number of elements in the array.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

# **CD3D11\_RENDER\_TARGET\_VIEW\_DESC::CD3D11\_RENDER\_TARGET\_VIEW\_DESC(ID3D11Texture2D\*,D3D11\_RTV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT)** **function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure that is initialized with 2D texture values.

## Syntax

C++

```
void CD3D11_RENDER_TARGET_VIEW_DESC(
    ID3D11Texture2D      *pTex2D,
    D3D11_RTV_DIMENSION  viewDimension,
    DXGI_FORMAT           format,
    UINT                  mipSlice,
    UINT                  firstArraySlice,
    UINT                  arraySize
);
```

## Parameters

`pTex2D`

A pointer to a `ID3D11Texture2D` interface for a 2D texture.

`viewDimension`

A `D3D11_RTV_DIMENSION`-typed value that specifies the resource type of the view.

`format`

A `DXGI_FORMAT`-typed value that specifies the viewing format.

`mipSlice`

The index of the mipmap level to use mip slice.

`firstArraySlice`

The index of the first element to use in an array of elements.

`arraySize`

Number of elements in the array.

## Return value

None

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

# **CD3D11\_RENDER\_TARGET\_VIEW\_DESC::CD3D11\_RENDER\_TARGET\_VIEW\_DESC(ID3D11Texture3D\*,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure that is initialized with 3D texture values.

## Syntax

C++

```
void CD3D11_RENDER_TARGET_VIEW_DESC(
    ID3D11Texture3D *pTex3D,
    DXGI_FORMAT      format,
    UINT             mipSlice,
    UINT             firstWSlice,
    UINT             wSize
);
```

## Parameters

`pTex3D`

A pointer to a `ID3D11Texture3D` interface for a 3D texture.

`format`

A `DXGI_FORMAT`-typed value that specifies the viewing format.

`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*.

## Return value

None

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_RENDER\_TARGET\_VIEW\_DESC::~CD3D11\_RENDER\_TARGET\_VIEW\_DESC function (d3d11.h)

Article 02/22/2024

Destroys an instance of a [CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_RENDER_TARGET_VIEW_DESC();
```

## Return value

None

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_RENDER\\_TARGET\\_VIEW\\_DESC](#)

# CD3D11\_VIEWPORT class

Article 12/01/2017

Represents a viewport and provides convenience methods for creating viewports.

## Members

The **CD3D11\_VIEWPORT** class inherits from [D3D11\\_VIEWPORT](#). **CD3D11\_VIEWPORT** also has these types of members:

- Constructors
- Methods

## Constructors

The **CD3D11\_VIEWPORT** class has these constructors.

[ ] Expand table

Constructor	Description
<a href="#">CD3D11_VIEWPORT()</a>	Instantiates a new instance of an uninitialized <b>CD3D11_VIEWPORT</b> structure.
<a href="#">CD3D11_VIEWPORT(D3D11_VIEWPORT_values) (FLOAT,FLOAT,FLOAT,FLOAT,FLOAT,FLOAT)</a>	Instantiates a new instance of a <b>CD3D11_VIEWPORT</b> structure that is initialized with <a href="#">D3D11_VIEWPORT</a> values.
<a href="#">CD3D11_VIEWPORT(D3D11_VIEWPORT_values2) (ID3D11Buffer, ID3D11RenderTargetView, FLOAT, FLOAT, FLOAT)</a>	Instantiates a new instance of a <b>CD3D11_VIEWPORT</b> structure that is initialized with <a href="#">D3D11_BUFFER_RTV</a> values.
<a href="#">CD3D11_VIEWPORT(D3D11_VIEWPORT_values3) (ID3D11Texture1D, ID3D11RenderTargetView, FLOAT, FLOAT, FLOAT)</a>	Instantiates a new instance of a <b>CD3D11_VIEWPORT</b> structure that is initialized with <a href="#">D3D11_TEX1D_RTV</a> or <a href="#">D3D11_TEX1D_ARRAY_RTV</a> values.

Constructor	Description
<code>CD3D11_VIEWPORT(D3D11_VIEWPORT_values4) (ID3D11Texture2D, ID3D11RenderTargetView, FLOAT, FLOAT, FLOAT, FLOAT)</code>	Instantiates a new instance of a <b>CD3D11_VIEWPORT</b> structure that is initialized with 2D texture values.
<code>CD3D11_VIEWPORT(D3D11_VIEWPORT_values5) (ID3D11Texture3D, ID3D11RenderTargetView, FLOAT, FLOAT, FLOAT, FLOAT)</code>	Instantiates a new instance of a <b>CD3D11_VIEWPORT</b> structure that is initialized with 3D texture values.

## Methods

The **CD3D11\_VIEWPORT** class has these methods.

[+] Expand table

Method	Description
<code>~CD3D11_VIEWPORT()</code>	Destroys an instance of a <b>CD3D11_VIEWPORT</b> structure.
<code>CD3D11_VIEWPORT(D3D11_VIEWPORT)(const &amp;D3D11_VIEWPORT)</code>	Instantiates a new instance of a <b>CD3D11_VIEWPORT</b> structure that is initialized with a <b>D3D11_VIEWPORT</b> structure.
<code>D3D11_VIEWPORT()</code>	This operator returns the address of a <b>D3D11_VIEWPORT</b> structure that contains the data from the <b>CD3D11_VIEWPORT</b> instance.

## Remarks

Here is how D3D11.h defines **CD3D11\_VIEWPORT**:

```
struct CD3D11_VIEWPORT : public D3D11_VIEWPORT
{
    CD3D11_VIEWPORT()
    {}
    explicit CD3D11_VIEWPORT( const D3D11_VIEWPORT& o ) :
```

```

D3D11_VIEWPORT( o )
{
explicit CD3D11_VIEWPORT(
    FLOAT topLeftX,
    FLOAT topLeftY,
    FLOAT width,
    FLOAT height,
    FLOAT minDepth = D3D11_MIN_DEPTH,
    FLOAT maxDepth = D3D11_MAX_DEPTH )
{
    TopLeftX = topLeftX;
    TopLeftY = topLeftY;
    Width = width;
    Height = height;
    MinDepth = minDepth;
    MaxDepth = maxDepth;
}
explicit CD3D11_VIEWPORT(
    _In_ ID3D11Buffer*,
    _In_ ID3D11RenderTargetView* pRTView,
    FLOAT topLeftX = 0.0f,
    FLOAT minDepth = D3D11_MIN_DEPTH,
    FLOAT maxDepth = D3D11_MAX_DEPTH )
{
    D3D11_RENDER_TARGET_VIEW_DESC RTVDesc;
    pRTView->GetDesc( &RTVDesc );
    UINT NumElements = 0;
    switch ( RTVDesc.ViewDimension )
    {
    case D3D11_RTV_DIMENSION_BUFFER:
        NumElements = RTVDesc.Buffer.NumElements;
        break;
    default: break;
    }
    TopLeftX = topLeftX;
    TopLeftY = 0.0f;
    Width = NumElements - topLeftX;
    Height = 1.0f;
    MinDepth = minDepth;
    MaxDepth = maxDepth;
}
explicit CD3D11_VIEWPORT(
    _In_ ID3D11Texture1D* pTex1D,
    _In_ ID3D11RenderTargetView* pRTView,
    FLOAT topLeftX = 0.0f,
    FLOAT minDepth = D3D11_MIN_DEPTH,
    FLOAT maxDepth = D3D11_MAX_DEPTH )
{
    D3D11_TEXTURE1D_DESC TexDesc;
    pTex1D->GetDesc( &TexDesc );
    D3D11_RENDER_TARGET_VIEW_DESC RTVDesc;
    pRTView->GetDesc( &RTVDesc );
    UINT MipSlice = 0;
    switch ( RTVDesc.ViewDimension )
    {

```

```

    case D3D11_RTV_DIMENSION_TEXTURE1D:
        MipSlice = RTVDesc.Texture1D.MipSlice;
        break;
    case D3D11_RTV_DIMENSION_TEXTURE1DARRAY:
        MipSlice = RTVDesc.Texture1DArray.MipSlice;
        break;
    default: break;
}
const UINT SubResourceWidth = TexDesc.Width / (UINT( 1 ) << MipSlice);
TopLeftX = topLeftX;
TopLeftY = 0.0f;
Width = (SubResourceWidth ? SubResourceWidth : 1) - topLeftX;
Height = 1.0f;
MinDepth = minDepth;
MaxDepth = maxDepth;
}
explicit CD3D11_VIEWPORT(
    _In_ ID3D11Texture2D* pTex2D,
    _In_ ID3D11RenderTargetView* pRTView,
    FLOAT topLeftX = 0.0f,
    FLOAT topLeftY = 0.0f,
    FLOAT minDepth = D3D11_MIN_DEPTH,
    FLOAT maxDepth = D3D11_MAX_DEPTH )
{
D3D11_TEXTURE2D_DESC TexDesc;
pTex2D->GetDesc( &TexDesc );
D3D11_RENDER_TARGET_VIEW_DESC RTVDesc;
pRTView->GetDesc( &RTVDesc );
UINT MipSlice = 0;
switch (RTVDesc.ViewDimension)
{
case D3D11_RTV_DIMENSION_TEXTURE2D:
    MipSlice = RTVDesc.Texture2D.MipSlice;
    break;
case D3D11_RTV_DIMENSION_TEXTURE2DARRAY:
    MipSlice = RTVDesc.Texture2DArray.MipSlice;
    break;
case D3D11_RTV_DIMENSION_TEXTURE2DMS:
case D3D11_RTV_DIMENSION_TEXTURE2DMSARRAY:
    break;
default: break;
}
const UINT SubResourceWidth = TexDesc.Width / (UINT( 1 ) << MipSlice);
const UINT SubResourceHeight = TexDesc.Height / (UINT( 1 ) << MipSlice);
TopLeftX = topLeftX;
TopLeftY = topLeftY;
Width = (SubResourceWidth ? SubResourceWidth : 1) - topLeftX;
Height = (SubResourceHeight ? SubResourceHeight : 1) - topLeftY;
MinDepth = minDepth;
MaxDepth = maxDepth;
}
explicit CD3D11_VIEWPORT(
    _In_ ID3D11Texture3D* pTex3D,
    _In_ ID3D11RenderTargetView* pRTView,
    FLOAT topLeftX = 0.0f,

```

```

FLOAT topLeftY = 0.0f,
FLOAT minDepth = D3D11_MIN_DEPTH,
FLOAT maxDepth = D3D11_MAX_DEPTH )
{
    D3D11_TEXTURE3D_DESC TexDesc;
    pTex3D->GetDesc( &TexDesc );
    D3D11_RENDER_TARGET_VIEW_DESC RTVDesc;
    pRTView->GetDesc( &RTVDesc );
    UINT MipSlice = 0;
    switch (RTVDesc.ViewDimension)
    {
        case D3D11_RT_V_DIMENSION_TEXTURE3D:
            MipSlice = RTVDesc.Texture3D.MipSlice;
            break;
        default: break;
    }
    const UINT SubResourceWidth = TexDesc.Width / (UINT( 1 ) << MipSlice);
    const UINT SubResourceHeight = TexDesc.Height / (UINT( 1 ) << MipSlice);
    TopLeftX = topLeftX;
    TopLeftY = topLeftY;
    Width = (SubResourceWidth ? SubResourceWidth : 1) - topLeftX;
    Height = (SubResourceHeight ? SubResourceHeight : 1) - topLeftY;
    MinDepth = minDepth;
    MaxDepth = maxDepth;
}
~CD3D11_VIEWPORT() {}
operator const D3D11_VIEWPORT&() const { return *this; }
};

```

## Requirements

[ ] [Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_VIEWPORT](#) ↗

[CD3D11 Helper Structures](#) ↗

# CD3D11\_VIEWPORT::CD3D11\_VIEWPORT function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_VIEWPORT](#) structure.

## Syntax

C++

```
void CD3D11_VIEWPORT();
```

## Return value

None

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_VIEWPORT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_VIEWPORT::CD3D11\_VIEWPORT( const D3D11\_VIEWPORT&) function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of a [CD3D11\\_VIEWPORT](#) structure that is initialized with a [D3D11\\_VIEWPORT](#) structure.

## Syntax

C++

```
void CD3D11_VIEWPORT(  
    [ref] const D3D11_VIEWPORT & o  
) ;
```

## Parameters

[ref] o

Type: [const D3D11\\_VIEWPORT](#)

Address of the [D3D11\\_VIEWPORT](#) structure that initializes the [CD3D11\\_VIEWPORT](#) structure.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_VIEWPORT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_VIEWPORT::CD3D11\_VIEWPORT( FLOAT,FLOAT,FLOAT,FLOAT,FLOAT,FLOAT) function (d3d11.h)

Article04/02/2021

Instantiates a new instance of a [CD3D11\\_VIEWPORT](#) structure that is initialized with [D3D11\\_VIEWPORT](#) values.

## Syntax

C++

```
void CD3D11_VIEWPORT(  
    FLOAT topLeftX,  
    FLOAT topLeftY,  
    FLOAT width,  
    FLOAT height,  
    FLOAT minDepth,  
    FLOAT maxDepth  
) ;
```

## Parameters

`topLeftX`

Type: [FLOAT](#)

X position of the left hand side of the viewport. Ranges between [D3D11\\_VIEWPORT\\_BOUNDS\\_MIN](#) and [D3D11\\_VIEWPORT\\_BOUNDS\\_MAX](#).

`topLeftY`

Type: [FLOAT](#)

Y position of the top of the viewport. Ranges between [D3D11\\_VIEWPORT\\_BOUNDS\\_MIN](#) and [D3D11\\_VIEWPORT\\_BOUNDS\\_MAX](#).

`width`

Type: [FLOAT](#)

Width of the viewport.

`height`

Type: **FLOAT**

Height of the viewport.

`minDepth`

Type: **FLOAT**

Minimum depth of the viewport. Ranges between 0 and 1.

`maxDepth`

Type: **FLOAT**

Maximum depth of the viewport. Ranges between 0 and 1.

## Return value

None

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_VIEWPORT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

---

---

Get help at Microsoft Q&A

# **CD3D11\_VIEWPORT::CD3D11\_VIEWPORT( ID3D11Texture2D\*,ID3D11RenderTargetV iew\*,FLOAT,FLOAT,FLOAT,FLOAT)**

## **function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_VIEWPORT](#) structure that is initialized with 2D texture values.

## Syntax

C++

```
void CD3D11_VIEWPORT(  
    ID3D11Texture2D      *pTex2D,  
    ID3D11RenderTargetView *pRTView,  
    FLOAT                 topLeftX,  
    FLOAT                 topLeftY,  
    FLOAT                 minDepth,  
    FLOAT                 maxDepth  
) ;
```

## Parameters

`pTex2D`

A pointer to a `ID3D11Texture2D` interface for a 2D texture.

`pRTView`

A pointer to a `ID3D11RenderTargetView` interface for the render-target view.

`topLeftX`

X position of the left hand side of the viewport. Ranges between `D3D11_VIEWPORT_BOUNDS_MIN` and `D3D11_VIEWPORT_BOUNDS_MAX`.

`topLeftY`

Y position of the top of the viewport. Ranges between D3D11\_VIEWPORT\_BOUNDS\_MIN and D3D11\_VIEWPORT\_BOUNDS\_MAX.

`minDepth`

Minimum depth of the viewport. Ranges between 0 and 1.

`maxDepth`

Maximum depth of the viewport. Ranges between 0 and 1.

## Return value

None

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_VIEWPORT](#)

---

## Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# **CD3D11\_VIEWPORT::CD3D11\_VIEWPORT( ID3D11Texture1D\*,ID3D11RenderTargetVi ew\*,FLOAT,FLOAT,FLOAT) function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_VIEWPORT](#) structure that is initialized with D3D11\_TEX1D\_RTV or D3D11\_TEX1D\_ARRAY\_RTV values.

## Syntax

C++

```
void CD3D11_VIEWPORT(  
    ID3D11Texture1D      *pTex1D,  
    ID3D11RenderTargetView *pRTView,  
    FLOAT                 topLeftX,  
    FLOAT                 minDepth,  
    FLOAT                 maxDepth  
>;
```

## Parameters

pTex1D

A pointer to a **ID3D11Texture1D** interface for a 1D texture.

pRTView

A pointer to a **ID3D11RenderTargetView** interface for the render-target view.

topLeftX

X position of the left hand side of the viewport. Ranges between D3D11\_VIEWPORT\_BOUNDS\_MIN and D3D11\_VIEWPORT\_BOUNDS\_MAX.

minDepth

Minimum depth of the viewport. Ranges between 0 and 1.

maxDepth

Maximum depth of the viewport. Ranges between 0 and 1.

## Return value

None

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_VIEWPORT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_VIEWPORT::CD3D11\_VIEWPORT( FLOAT,FLOAT,FLOAT,FLOAT,FLOAT,FLOAT) function (d3d11.h)

Article04/02/2021

Instantiates a new instance of a [CD3D11\\_VIEWPORT](#) structure that is initialized with [D3D11\\_VIEWPORT](#) values.

## Syntax

C++

```
void CD3D11_VIEWPORT(  
    FLOAT topLeftX,  
    FLOAT topLeftY,  
    FLOAT width,  
    FLOAT height,  
    FLOAT minDepth,  
    FLOAT maxDepth  
) ;
```

## Parameters

`topLeftX`

Type: [FLOAT](#)

X position of the left hand side of the viewport. Ranges between [D3D11\\_VIEWPORT\\_BOUNDS\\_MIN](#) and [D3D11\\_VIEWPORT\\_BOUNDS\\_MAX](#).

`topLeftY`

Type: [FLOAT](#)

Y position of the top of the viewport. Ranges between [D3D11\\_VIEWPORT\\_BOUNDS\\_MIN](#) and [D3D11\\_VIEWPORT\\_BOUNDS\\_MAX](#).

`width`

Type: [FLOAT](#)

Width of the viewport.

`height`

Type: **FLOAT**

Height of the viewport.

`minDepth`

Type: **FLOAT**

Minimum depth of the viewport. Ranges between 0 and 1.

`maxDepth`

Type: **FLOAT**

Maximum depth of the viewport. Ranges between 0 and 1.

## Return value

None

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_VIEWPORT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# CD3D11\_VIEWPORT::CD3D11\_VIEWPORT( FLOAT,FLOAT,FLOAT,FLOAT,FLOAT,FLOAT) function (d3d11.h)

Article04/02/2021

Instantiates a new instance of a [CD3D11\\_VIEWPORT](#) structure that is initialized with [D3D11\\_VIEWPORT](#) values.

## Syntax

C++

```
void CD3D11_VIEWPORT(  
    FLOAT topLeftX,  
    FLOAT topLeftY,  
    FLOAT width,  
    FLOAT height,  
    FLOAT minDepth,  
    FLOAT maxDepth  
) ;
```

## Parameters

`topLeftX`

Type: [FLOAT](#)

X position of the left hand side of the viewport. Ranges between [D3D11\\_VIEWPORT\\_BOUNDS\\_MIN](#) and [D3D11\\_VIEWPORT\\_BOUNDS\\_MAX](#).

`topLeftY`

Type: [FLOAT](#)

Y position of the top of the viewport. Ranges between [D3D11\\_VIEWPORT\\_BOUNDS\\_MIN](#) and [D3D11\\_VIEWPORT\\_BOUNDS\\_MAX](#).

`width`

Type: [FLOAT](#)

Width of the viewport.

`height`

Type: **FLOAT**

Height of the viewport.

`minDepth`

Type: **FLOAT**

Minimum depth of the viewport. Ranges between 0 and 1.

`maxDepth`

Type: **FLOAT**

Maximum depth of the viewport. Ranges between 0 and 1.

## Return value

None

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_VIEWPORT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

---

---

Get help at Microsoft Q&A

# CD3D11\_VIEWPORT::~CD3D11\_VIEWPORT function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_VIEWPORT](#) structure.

## Syntax

C++

```
void ~CD3D11_VIEWPORT();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_VIEWPORT](#)

# CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC class

Article 12/01/2017

Represents a depth-stencil view and provides convenience methods for creating depth-stencil views.

## Members

The CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC class inherits from [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#).

CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC class has these constructors.

  Expand table

Constructor	Description
<a href="#">CD3D11_DEPTH_STENCIL_VIEW_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_DEPTH_STENCIL_VIEW_DESC structure.
<a href="#">CD3D11_DEPTH_STENCIL_VIEW_DESC(D3D11_DEPTH_STENCIL_VIEW_DESC_values)</a> <a href="#">(D3D11_DSV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_DEPTH_STENCIL_VIEW_DESC structure that is initialized with D3D11_DEPTH_STENCIL_VIEW_DESC values.
<a href="#">CD3D11_DEPTH_STENCIL_VIEW_DESC(D3D11_DEPTH_STENCIL_VIEW_DESC_values2)</a> <a href="#">(ID3D11Texture1D,D3D11_DSV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_DEPTH_STENCIL_VIEW_DESC structure that is initialized with D3D11_TEX1D_DSV or D3D11_TEX1D_ARRAY_DSV values.
<a href="#">CD3D11_DEPTH_STENCIL_VIEW_DESC(D3D11_DEPTH_STENCIL_VIEW_DESC_values3)</a> <a href="#">(ID3D11Texture2D,D3D11_DSV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT,UINT)</a>	Instantiates a new instance of a CD3D11_DEPTH_STENCIL_VIEW_DESC structure that is initialized with 2D texture values.

## Methods

The CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC class has these methods.

  Expand table

Method	Description
<a href="#">~CD3D11_DEPTH_STENCIL_VIEW_DESC()</a>	Destroys an instance of a CD3D11_DEPTH_STENCIL_VIEW_DESC structure.
<a href="#">CD3D11_DEPTH_STENCIL_VIEW_DESC(D3D11_DEPTH_STENCIL_VIEW_DESC)</a> <a href="#">(const &amp;D3D11_DEPTH_STENCIL_VIEW_DESC)</a>	Instantiates a new instance of a CD3D11_DEPTH_STENCIL_VIEW_DESC structure that is initialized with a D3D11_DEPTH_STENCIL_VIEW_DESC structure.
<a href="#">D3D11_DEPTH_STENCIL_VIEW_DESC()</a>	This operator returns the address of a D3D11_DEPTH_STENCIL_VIEW_DESC structure that contains the data from the CD3D11_DEPTH_STENCIL_VIEW_DESC instance.

## Remarks

Here is how D3D11.h defines CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC:

```
struct CD3D11_DEPTH_STENCIL_VIEW_DESC : public D3D11_DEPTH_STENCIL_VIEW_DESC
{
    CD3D11_DEPTH_STENCIL_VIEW_DESC()
    {}
    explicit CD3D11_DEPTH_STENCIL_VIEW_DESC( const D3D11_DEPTH_STENCIL_VIEW_DESC& o ) :
        D3D11_DEPTH_STENCIL_VIEW_DESC( o )
    {}
    explicit CD3D11_DEPTH_STENCIL_VIEW_DESC(
        D3D11_DSV_DIMENSION viewDimension,
        DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
        UINT mipSlice = 0,
        UINT firstArraySlice = 0,
        UINT arraySize = -1,
        UINT flags = 0 )
    {
        Format = format;
        ViewDimension = viewDimension;
        Flags = flags;
        switch (viewDimension)
        {
            case D3D11_DSV_DIMENSION_TEXTURE1D:
                Texture1D.MipSlice = mipSlice;
                break;
            case D3D11_DSV_DIMENSION_TEXTURE1DARRAY:
                Texture1DArray.MipSlice = mipSlice;
                Texture1DArray.FirstArraySlice = firstArraySlice;
                Texture1DArray.ArraySize = arraySize;
                break;
            case D3D11_DSV_DIMENSION_TEXTURE2D:
                Texture2D.MipSlice = mipSlice;
                break;
        }
    }
};
```

```

        case D3D11_DSV_DIMENSION_TEXTURE2DARRAY:
            Texture2DArray.MipSlice = mipSlice;
            Texture2DArray.FirstArraySlice = firstArraySlice;
            Texture2DArray.ArraySize = arraySize;
            break;
        case D3D11_DSV_DIMENSION_TEXTURE2DMS:
            break;
        case D3D11_DSV_DIMENSION_TEXTURE2DMSARRAY:
            Texture2DMSArray.FirstArraySlice = firstArraySlice;
            Texture2DMSArray.ArraySize = arraySize;
            break;
        default: break;
    }
}

explicit CD3D11_DEPTH_STENCIL_VIEW_DESC(
    _In_ ID3D11Texture1D* pTex1D,
    D3D11_DSV_DIMENSION viewDimension,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mipSlice = 0,
    UINT firstArraySlice = 0,
    UINT arraySize = -1,
    UINT flags = 0 )
{
    ViewDimension = viewDimension;
    Flags = flags;
    if (DXGI_FORMAT_UNKNOWN == format ||
        (-1 == arraySize && D3D11_DSV_DIMENSION_TEXTURE1DARRAY == viewDimension))
    {
        D3D11_TEXTURE1D_DESC TexDesc;
        pTex1D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == arraySize) arraySize = TexDesc.ArraySize - firstArraySlice;
    }
    Format = format;
    switch (viewDimension)
    {
        case D3D11_DSV_DIMENSION_TEXTURE1D:
            Texture1D.MipSlice = mipSlice;
            break;
        case D3D11_DSV_DIMENSION_TEXTURE1DARRAY:
            Texture1DArray.MipSlice = mipSlice;
            Texture1DArray.FirstArraySlice = firstArraySlice;
            Texture1DArray.ArraySize = arraySize;
            break;
        default: break;
    }
}

explicit CD3D11_DEPTH_STENCIL_VIEW_DESC(
    _In_ ID3D11Texture2D* pTex2D,
    D3D11_DSV_DIMENSION viewDimension,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mipSlice = 0,
    UINT firstArraySlice = 0,
    UINT arraySize = -1,
    UINT flags = 0 )
{
    ViewDimension = viewDimension;
    Flags = flags;
    if (DXGI_FORMAT_UNKNOWN == format ||
        (-1 == arraySize &&
            (D3D11_DSV_DIMENSION_TEXTURE2DARRAY == viewDimension ||
            D3D11_DSV_DIMENSION_TEXTURE2DMSARRAY == viewDimension)))

```

```

{
    D3D11_TEXTURE2D_DESC TexDesc;
    pTex2D->GetDesc( &TexDesc );
    if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
    if (-1 == arraySize) arraySize = TexDesc.ArraySize - firstArraySlice;
}
Format = format;
switch (viewDimension)
{
case D3D11_DSV_DIMENSION_TEXTURE2D:
    Texture2D.MipSlice = mipSlice;
    break;
case D3D11_DSV_DIMENSION_TEXTURE2DARRAY:
    Texture2DArray.MipSlice = mipSlice;
    Texture2DArray.FirstArraySlice = firstArraySlice;
    Texture2DArray.ArraySize = arraySize;
    break;
case D3D11_DSV_DIMENSION_TEXTURE2DMS:
    break;
case D3D11_DSV_DIMENSION_TEXTURE2DMSARRAY:
    Texture2DMSArray.FirstArraySlice = firstArraySlice;
    Texture2DMSArray.ArraySize = arraySize;
    break;
default: break;
}
}
~CD3D11_DEPTH_STENCIL_VIEW_DESC() {}
operator const D3D11_DEPTH_STENCIL_VIEW_DESC&() const { return *this; }
};

```

## Requirements

[ ] [Expand table](#)

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) ↗

[CD3D11 Helper Structures](#) ↗

# CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC::CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC function (d3d11.h)

Article02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_DEPTH_STENCIL_VIEW_DESC();
```

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC::CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC(const CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure that is initialized with a [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_DEPTH_STENCIL_VIEW_DESC(
    [ref] const D3D11_DEPTH_STENCIL_VIEW_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)

Address of the [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure that initializes the [CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure.

## Return value

None

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# **CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC::CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC(D3D11\_DSV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure that is initialized with [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_DEPTH_STENCIL_VIEW_DESC(
    D3D11_DSV_DIMENSION viewDimension,
    DXGI_FORMAT          format,
    UINT                 mipSlice,
    UINT                 firstArraySlice,
    UINT                 arraySize,
    UINT                 flags
);
```

## Parameters

`viewDimension`

Type: [D3D11\\_DSV\\_DIMENSION](#)

A [D3D11\\_DSV\\_DIMENSION](#)-typed value that specifies the depth-stencil type of the view.

`format`

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#)-typed value that specifies the viewing format.

`mipSlice`

Type: [UINT](#)

The index of the mipmap level to use mip slice.

`firstArraySlice`

Type: [UINT](#)

The index of the first element to use in an array of elements.

`arraySize`

Type: [UINT](#)

Number of elements in the array.

`flags`

Type: [UINT](#)

A value that describes whether the texture is read only. Pass 0 to specify that it is not read only; otherwise, pass one of the members of the [D3D11\\_DSV\\_FLAG](#) enumerated type.

## Return value

None

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# **CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC::CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC(ID3D11Texture2D\*,D3D11\_DSV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT,UINT)**

## **function (d3d11.h)**

Article08/31/2022

Instantiates a new instance of a [CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure that is initialized with 2D texture values.

## Syntax

C++

```
void CD3D11_DEPTH_STENCIL_VIEW_DESC(
    ID3D11Texture2D      *pTex2D,
    D3D11_DSV_DIMENSION viewDimension,
    DXGI_FORMAT           format,
    UINT                  mipSlice,
    UINT                  firstArraySlice,
    UINT                  arraySize,
    UINT                  flags
);
```

## Parameters

`pTex2D`

A pointer to a `ID3D11Texture2D`.

`viewDimension`

A `D3D11_DSV_DIMENSION`-typed value that specifies the depth-stencil type of the view.

`format`

A `DXGI_FORMAT`-typed value that specifies the viewing format.

`mipSlice`

The index of the mipmap level to use mip slice.

`firstArraySlice`

The index of the first element to use in an array of elements.

`arraySize`

Number of elements in the array.

`flags`

A value that describes whether the texture is read only. Pass 0 to specify that it is not read only; otherwise, pass one of the members of the **D3D11\_DSV\_FLAG** enumerated type.

## Return value

None

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC::CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC(D3D11\_DSV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure that is initialized with [D3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_DEPTH_STENCIL_VIEW_DESC(
    D3D11_DSV_DIMENSION viewDimension,
    DXGI_FORMAT          format,
    UINT                 mipSlice,
    UINT                 firstArraySlice,
    UINT                 arraySize,
    UINT                 flags
);
```

## Parameters

`viewDimension`

Type: [D3D11\\_DSV\\_DIMENSION](#)

A [D3D11\\_DSV\\_DIMENSION](#)-typed value that specifies the depth-stencil type of the view.

`format`

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#)-typed value that specifies the viewing format.

`mipSlice`

Type: [UINT](#)

The index of the mipmap level to use mip slice.

`firstArraySlice`

Type: [UINT](#)

The index of the first element to use in an array of elements.

`arraySize`

Type: [UINT](#)

Number of elements in the array.

`flags`

Type: [UINT](#)

A value that describes whether the texture is read only. Pass 0 to specify that it is not read only; otherwise, pass one of the members of the [D3D11\\_DSV\\_FLAG](#) enumerated type.

## Return value

None

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC::~CD3D11\_DEPTH\_STENCIL\_VIEW\_DESC function (d3d11.h)

Article 02/22/2024

Destroys an instance of a [CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_DEPTH_STENCIL_VIEW_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_DEPTH\\_STENCIL\\_VIEW\\_DESC](#)

# CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC class

Article12/01/2017

Represents a unordered-access view and provides convenience methods for creating unordered-access views.

## Members

The CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC class inherits from [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#).

CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC class has these constructors.

[Expand table](#)

Constructor	Description
<a href="#">CD3D11_UNORDERED_ACCESS_VIEW_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_UNORDERED_ACCESS_VIEW_DESC structure.
<a href="#">CD3D11_UNORDERED_ACCESS_VIEW_DESC(D3D11_UNORDERED_ACCESS_VIEW_DESC_values)</a> (D3D11_UAV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT,UINT)	Instantiates a new instance of a CD3D11_UNORDERED_ACCESS_VIEW_DESC structure that is initialized with <a href="#">D3D11_UNORDERED_ACCESS_VIEW_DESC</a> values.
<a href="#">CD3D11_UNORDERED_ACCESS_VIEW_DESC(D3D11_UNORDERED_ACCESS_VIEW_DESC_values2)</a> (ID3D11Buffer,DXGI_FORMAT,UINT,UINT,UINT)	Instantiates a new instance of a CD3D11_UNORDERED_ACCESS_VIEW_DESC structure that is initialized with <a href="#">D3D11_BUFFER_UAV</a> values.
<a href="#">CD3D11_UNORDERED_ACCESS_VIEW_DESC(D3D11_UNORDERED_ACCESS_VIEW_DESC_values3)</a> (ID3D11Texture1D,D3D11_UAV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT)	Instantiates a new instance of a CD3D11_UNORDERED_ACCESS_VIEW_DESC structure that is initialized with <a href="#">D3D11_TEX1D_UAV</a> or <a href="#">D3D11_TEX1D_ARRAY_UAV</a> values.
<a href="#">CD3D11_UNORDERED_ACCESS_VIEW_DESC(D3D11_UNORDERED_ACCESS_VIEW_DESC_values4)</a> (ID3D11Texture2D,D3D11_UAV_DIMENSION,DXGI_FORMAT,UINT,UINT,UINT)	Instantiates a new instance of a CD3D11_UNORDERED_ACCESS_VIEW_DESC structure that is initialized with 2D texture values.
<a href="#">CD3D11_UNORDERED_ACCESS_VIEW_DESC(D3D11_UNORDERED_ACCESS_VIEW_DESC_values5)</a> (ID3D11Texture3D,DXGI_FORMAT,UINT,UINT,UINT)	Instantiates a new instance of a CD3D11_UNORDERED_ACCESS_VIEW_DESC structure that is initialized with 3D texture values.

## Methods

The CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC class has these methods.

Method	Description
<a href="#">~CD3D11_UNORDERED_ACCESS_VIEW_DESC()</a>	Destroys an instance of a CD3D11_UNORDERED_ACCESS_VIEW_DESC structure.
<a href="#">CD3D11_UNORDERED_ACCESS_VIEW_DESC(D3D11_UNORDERED_ACCESS_VIEW_DESC)</a> <a href="#">(const &amp;D3D11_UNORDERED_ACCESS_VIEW_DESC)</a>	Instantiates a new instance of a CD3D11_UNORDERED_ACCESS_VIEW_DESC structure that is initialized with a D3D11_UNORDERED_ACCESS_VIEW_DESC structure.
<a href="#">D3D11_UNORDERED_ACCESS_VIEW_DESC()</a>	This operator returns the address of a D3D11_UNORDERED_ACCESS_VIEW_DESC structure that contains the data from the CD3D11_UNORDERED_ACCESS_VIEW_DESC instance.

## Remarks

Here is how D3D11.h defines CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC:

```
struct CD3D11_UNORDERED_ACCESS_VIEW_DESC : public D3D11_UNORDERED_ACCESS_VIEW_DESC
{
    CD3D11_UNORDERED_ACCESS_VIEW_DESC()
    {}
    explicit CD3D11_UNORDERED_ACCESS_VIEW_DESC( const D3D11_UNORDERED_ACCESS_VIEW_DESC& o ) :
        D3D11_UNORDERED_ACCESS_VIEW_DESC( o )
    {}
    explicit CD3D11_UNORDERED_ACCESS_VIEW_DESC(
        D3D11_UAV_DIMENSION viewDimension,
        DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
        UINT mipSlice = 0, // FirstElement for BUFFER
        UINT firstArraySlice = 0, // NumElements for BUFFER, FirstWSlice for TEXTURE3D
        UINT arraySize = -1, // WSize for TEXTURE3D
        UINT flags = 0 ) // BUFFER only
    {
        Format = format;
        ViewDimension = viewDimension;
        switch (viewDimension)
        {
            case D3D11_UAV_DIMENSION_BUFFER:
                Buffer.FirstElement = mipSlice;
                Buffer.NumElements = firstArraySlice;
                Buffer.Flags = flags;
                break;
            case D3D11_UAV_DIMENSION_TEXTURE1D:
                Texture1D.MipSlice = mipSlice;
                break;
            case D3D11_UAV_DIMENSION_TEXTURE1DARRAY:
                Texture1DArray.MipSlice = mipSlice;
                Texture1DArray.FirstArraySlice = firstArraySlice;
                Texture1DArray.ArraySize = arraySize;
                break;
            case D3D11_UAV_DIMENSION_TEXTURE2D:
                Texture2D.MipSlice = mipSlice;
                break;
            case D3D11_UAV_DIMENSION_TEXTURE2DARRAY:
                Texture2DArray.MipSlice = mipSlice;
                break;
        }
    }
};
```

```

        Texture2DArray.FirstArraySlice = firstArraySlice;
        Texture2DArray.ArraySize = arraySize;
        break;
    case D3D11_UAV_DIMENSION_TEXTURE3D:
        Texture3D.MipSlice = mipSlice;
        Texture3D.FirstWSlice = firstArraySlice;
        Texture3D.WSize = arraySize;
        break;
    default: break;
}
}

explicit CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    _In_ ID3D11Buffer*,
    DXGI_FORMAT format,
    UINT firstElement,
    UINT numElements,
    UINT flags = 0 )
{
    Format = format;
    ViewDimension = D3D11_UAV_DIMENSION_BUFFER;
    Buffer.FirstElement = firstElement;
    Buffer.NumElements = numElements;
    Buffer.Flags = flags;
}

explicit CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    _In_ ID3D11Texture1D* pTex1D,
    D3D11_UAV_DIMENSION viewDimension,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mipSlice = 0,
    UINT firstArraySlice = 0,
    UINT arraySize = -1 )
{
    ViewDimension = viewDimension;
    if (DXGI_FORMAT_UNKNOWN == format ||
        (-1 == arraySize && D3D11_UAV_DIMENSION_TEXTURE1DARRAY == viewDimension))
    {
        D3D11_TEXTURE1D_DESC TexDesc;
        pTex1D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == arraySize) arraySize = TexDesc.ArraySize - firstArraySlice;
    }
    Format = format;
    switch (viewDimension)
    {
        case D3D11_UAV_DIMENSION_TEXTURE1D:
            Texture1D.MipSlice = mipSlice;
            break;
        case D3D11_UAV_DIMENSION_TEXTURE1DARRAY:
            Texture1DArray.MipSlice = mipSlice;
            Texture1DArray.FirstArraySlice = firstArraySlice;
            Texture1DArray.ArraySize = arraySize;
            break;
        default: break;
    }
}

explicit CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    _In_ ID3D11Texture2D* pTex2D,
    D3D11_UAV_DIMENSION viewDimension,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mipSlice = 0,
    UINT firstArraySlice = 0,
    UINT arraySize = -1 )
{
    ViewDimension = viewDimension;
    if (DXGI_FORMAT_UNKNOWN == format ||
        (-1 == arraySize && D3D11_UAV_DIMENSION_TEXTURE2DARRAY == viewDimension))
    {
        D3D11_TEXTURE2D_DESC TexDesc;
        pTex2D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
    }
}

```

```

        if (-1 == arraySize) arraySize = TexDesc.ArraySize - firstArraySlice;
    }
    Format = format;
    switch (viewDimension)
    {
        case D3D11_UAV_DIMENSION_TEXTURE2D:
            Texture2D.MipSlice = mipSlice;
            break;
        case D3D11_UAV_DIMENSION_TEXTURE2DARRAY:
            Texture2DArray.MipSlice = mipSlice;
            Texture2DArray.FirstArraySlice = firstArraySlice;
            Texture2DArray.ArraySize = arraySize;
            break;
        default: break;
    }
}
explicit CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    _In_ ID3D11Texture3D* pTex3D,
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN,
    UINT mipSlice = 0,
    UINT firstWSlice = 0,
    UINT wSize = -1 )
{
    ViewDimension = D3D11_UAV_DIMENSION_TEXTURE3D;
    if (DXGI_FORMAT_UNKNOWN == format || -1 == wSize)
    {
        D3D11_TEXTURE3D_DESC TexDesc;
        pTex3D->GetDesc( &TexDesc );
        if (DXGI_FORMAT_UNKNOWN == format) format = TexDesc.Format;
        if (-1 == wSize) wSize = TexDesc.Depth - firstWSlice;
    }
    Format = format;
    Texture3D.MipSlice = mipSlice;
    Texture3D.FirstWSlice = firstWSlice;
    Texture3D.WSize = wSize;
}
~CD3D11_UNORDERED_ACCESS_VIEW_DESC() {}
operator const D3D11_UNORDERED_ACCESS_VIEW_DESC&() const { return *this; }
};

```

## Requirements

[Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

[CD3D11 Helper Structures](#)

# **CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC** function (d3d11.h)

Article02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC();
```

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC::CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC(const CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure that is initialized with a [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    [ref] const D3D11_UNORDERED_ACCESS_VIEW_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

Address of the [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure that initializes the [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure.

## Return value

None

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC::CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC(D3D11\_UAV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure that is initialized with [D3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) values.

## Syntax

C++

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    D3D11_UAV_DIMENSION viewDimension,
    DXGI_FORMAT          format,
    UINT                 mipSlice,
    UINT                 firstArraySlice,
    UINT                 arraySize,
    UINT                 flags
);
```

## Parameters

`viewDimension`

Type: [D3D11\\_UAV\\_DIMENSION](#)

A [D3D11\\_UAV\\_DIMENSION](#)-typed value that specifies the resource type of the view.

`format`

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#)-typed value that specifies the viewing format.

`mipSlice`

Type: [UINT](#)

The index of the mipmap level to use mip slice. **FirstElement** for [D3D11\\_BUFFER\\_UAV](#).

`firstArraySlice`

Type: [UINT](#)

The index of the first element to use in an array of elements.

**NumElements** for [D3D11\\_BUFFER\\_UAV](#). **FirstWSlice** for [D3D11\\_TEX3D\\_UAV](#).

`arraySize`

Type: [UINT](#)

Number of elements in the array. **WSize** for [D3D11\\_TEX3D\\_UAV](#).

`flags`

Type: [UINT](#)

A [D3D11\\_BUFFER\\_UAV\\_FLAG](#)-typed value that identifies view options for a buffer. For [D3D11\\_BUFFER\\_UAV](#) only.

## Return value

None

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# **CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC::CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC(ID3D11Buffer\*,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure that is initialized with [D3D11\\_BUFFER\\_UAV](#) values.

## Syntax

C++

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    ID3D11Buffer *unnamedParam1,
    DXGI_FORMAT    format,
    UINT           firstElement,
    UINT           numElements,
    UINT           flags
);
```

## Parameters

`unnamedParam1`

A pointer to a [ID3D11Buffer](#) interface for a buffer.

`format`

A [DXGI\\_FORMAT](#)-typed value that specifies the viewing format.

`firstElement`

Number of bytes between the beginning of the buffer and the first element to access.

`numElements`

Total number of elements in the view.

`flags`

A D3D11\_BUFFER\_UAV\_FLAG-typed value that identifies view options for a buffer.

## Return value

None

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# **CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC::CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC(ID3D11Texture1D\*,D3D11\_UAV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure that is initialized with [D3D11\\_TEX1D\\_UAV](#) or [D3D11\\_TEX1D\\_ARRAY\\_UAV](#) values.

## Syntax

C++

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    ID3D11Texture1D      *pTex1D,
    D3D11_UAV_DIMENSION viewDimension,
    DXGI_FORMAT           format,
    UINT                  mipSlice,
    UINT                  firstArraySlice,
    UINT                  arraySize
);
```

## Parameters

`pTex1D`

A pointer to a [ID3D11Texture1D](#) interface for a 1D texture.

`viewDimension`

A [D3D11\\_UAV\\_DIMENSION](#)-typed value that specifies the resource type of the view.

`format`

A [DXGI\\_FORMAT](#)-typed value that specifies the viewing format.

`mipSlice`

The index of the mipmap level to use mip slice.

`firstArraySlice`

The index of the first element to use in an array of elements.

`arraySize`

Number of elements in the array.

## Return value

None

## Requirements

[ Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# **CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC::CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC(ID3D11Texture2D\*,D3D11\_UAV\_DIMENSION,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)**

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure that is initialized with 2D texture values.

## Syntax

C++

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    ID3D11Texture2D      *pTex2D,
    D3D11_UAV_DIMENSION  viewDimension,
    DXGI_FORMAT           format,
    UINT                  mipSlice,
    UINT                  firstArraySlice,
    UINT                  arraySize
);
```

## Parameters

`pTex2D`

A pointer to a `ID3D11Texture2D` interface for a 2D texture.

`viewDimension`

A `D3D11_UAV_DIMENSION`-typed value that specifies the resource type of the view.

`format`

A `DXGI_FORMAT`-typed value that specifies the viewing format.

`mipSlice`

The index of the mipmap level to use mip slice.

`firstArraySlice`

The index of the first element to use in an array of elements.

`arraySize`

Number of elements in the array.

## Return value

None

## Requirements

[ Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# **CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC::CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC(ID3D11Texture3D\*,DXGI\_FORMAT,UINT,UINT,UINT) function (d3d11.h)**

Article08/31/2022

Instantiates a new instance of a [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure that is initialized with 3D texture values.

## Syntax

C++

```
void CD3D11_UNORDERED_ACCESS_VIEW_DESC(
    ID3D11Texture3D *pTex3D,
    DXGI_FORMAT      format,
    UINT             mipSlice,
    UINT             firstWSlice,
    UINT             wSize
);
```

## Parameters

`pTex3D`

A pointer to a `ID3D11Texture3D` interface for a 3D texture.

`format`

A `DXGI_FORMAT`-typed value that specifies the viewing format.

`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*.

## Return value

None

## Requirements

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC::~CD3D11\_UNORDERED\_ACCESS\_VIEW\_DESC C function (d3d11.h)

Article 02/22/2024

Destroys an instance of a [CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_UNORDERED_ACCESS_VIEW_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_UNORDERED\\_ACCESS\\_VIEW\\_DESC](#)

# CD3D11\_SAMPLER\_DESC class

Article 12/01/2017

Represents a sampler state and provides convenience methods for creating sampler states.

## Members

The CD3D11\_SAMPLER\_DESC class inherits from [D3D11\\_SAMPLER\\_DESC](#).

CD3D11\_SAMPLER\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_SAMPLER\_DESC class has these constructors.

[ ] [Expand table](#)

Constructor	Description
<a href="#">CD3D11_SAMPLER_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_SAMPLER_DESC structure.
<a href="#">CD3D11_SAMPLER_DESC(D3D11_SAMPLER_DESC_values)</a> <a href="#">(CD3D11_DEFAULT)</a>	Instantiates a new instance of a CD3D11_SAMPLER_DESC structure that is initialized with default sampler-state values.

## Methods

The CD3D11\_SAMPLER\_DESC class has these methods.

[ ] [Expand table](#)

Method	Description
<a href="#">~CD3D11_SAMPLER_DESC()</a>	Destroys an instance of a CD3D11_SAMPLER_DESC structure.

Method	Description
<a href="#">CD3D11_SAMPLER_DESC(D3D11_SAMPLER_DESC)(const &amp;D3D11_SAMPLER_DESC)</a>	Instantiates a new instance of a <b>D3D11_SAMPLER_DESC</b> structure that is initialized with a <b>D3D11_SAMPLER_DESC</b> structure.
<a href="#">D3D11_SAMPLER_DESC()</a>	This operator returns the address of a <b>D3D11_SAMPLER_DESC</b> structure that contains the data from the <b>CD3D11_SAMPLER_DESC</b> instance.

## Remarks

Here is how D3D11.h defines **CD3D11\_SAMPLER\_DESC**:

```
struct CD3D11_SAMPLER_DESC : public D3D11_SAMPLER_DESC
{
    CD3D11_SAMPLER_DESC()
    {}
    explicit CD3D11_SAMPLER_DESC( const D3D11_SAMPLER_DESC& o ) :
        D3D11_SAMPLER_DESC( o )
    {}
    explicit CD3D11_SAMPLER_DESC( CD3D11_DEFAULT )
    {
        Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
        AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;
        AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;
        AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;
        MipLODBias = 0;
        MaxAnisotropy = 1;
        ComparisonFunc = D3D11_COMPARISON_NEVER;
        BorderColor[ 0 ] = 1.0f;
        BorderColor[ 1 ] = 1.0f;
        BorderColor[ 2 ] = 1.0f;
        BorderColor[ 3 ] = 1.0f;
        MinLOD = -3.402823466e+38F; // -FLT_MAX
        MaxLOD = 3.402823466e+38F; // FLT_MAX
    }
    explicit CD3D11_SAMPLER_DESC(
        D3D11_FILTER filter,
        D3D11_TEXTURE_ADDRESS_MODE addressU,
        D3D11_TEXTURE_ADDRESS_MODE addressV,
        D3D11_TEXTURE_ADDRESS_MODE addressW,
        FLOAT mipLODBias,
        UINT maxAnisotropy,
    );
};
```

```

D3D11_COMPARISON_FUNC comparisonFunc,
_In_reads_opt_( 4 ) const FLOAT* borderColor, // RGBA
FLOAT minLOD,
FLOAT maxLOD )
{
    Filter = filter;
    AddressU = addressU;
    AddressV = addressV;
    AddressW = addressW;
    MipLODBias = mipLODBias;
    MaxAnisotropy = maxAnisotropy;
    ComparisonFunc = comparisonFunc;
    const float defaultColor[ 4 ] = { 1.0f, 1.0f, 1.0f, 1.0f };
    if (!borderColor) borderColor = defaultColor;
    BorderColor[ 0 ] = borderColor[ 0 ];
    BorderColor[ 1 ] = borderColor[ 1 ];
    BorderColor[ 2 ] = borderColor[ 2 ];
    BorderColor[ 3 ] = borderColor[ 3 ];
    MinLOD = minLOD;
    MaxLOD = maxLOD;
}
~CD3D11_SAMPLER_DESC() {}
operator const D3D11_SAMPLER_DESC&() const { return *this; }
};

```

## Requirements

[Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_SAMPLER\\_DESC](#)

[CD3D11 Helper Structures](#)

# CD3D11\_SAMPLER\_DESC::CD3D11\_SAMPLER\_DESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_SAMPLER\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_SAMPLER_DESC();
```

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SAMPLER\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_SAMPLER\_DESC::CD3D11\_SAMPLER\_DESC(const D3D11\_SAMPLER\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_SAMPLER\\_DESC](#) structure that is initialized with a [D3D11\\_SAMPLER\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_SAMPLER_DESC(  
    [ref] const D3D11_SAMPLER_DESC & o  
) ;
```

## Parameters

[ref] o

Type: [const D3D11\\_SAMPLER\\_DESC](#)

Address of the [D3D11\\_SAMPLER\\_DESC](#) structure that initializes the [CD3D11\\_SAMPLER\\_DESC](#) structure.

## Return value

None

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SAMPLER\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_SAMPLER\_DESC::CD3D11\_SAMPLER\_DESC(CD3D11\_DEFAULT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_SAMPLER\\_DESC](#) structure that is initialized with default sampler-state values.

## Syntax

C++

```
void CD3D11_SAMPLER_DESC(  
    CD3D11_DEFAULT unnamedParam1  
) ;
```

## Parameters

unnamedParam1

Default sampler-state values.

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h

Requirement	Value
Library	D3D11.lib

## See also

[CD3D11\\_SAMPLER\\_DESC](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_SAMPLER\_DESC::~CD3D11\_SAMPLER\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_SAMPLER\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_SAMPLER_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_SAMPLER\\_DESC](#)

# CD3D11\_QUERY\_DESC class

Article 12/01/2017

Represents a query and provides convenience methods for creating queries.

## Members

The **CD3D11\_QUERY\_DESC** class inherits from [D3D11\\_QUERY\\_DESC](#). **CD3D11\_QUERY\_DESC** also has these types of members:

- Constructors
- Methods

## Constructors

The **CD3D11\_QUERY\_DESC** class has these constructors.

[Expand table](#)

Constructor	Description
<a href="#">CD3D11_QUERY_DESC()</a>	Instantiates a new instance of an uninitialized <b>CD3D11_QUERY_DESC</b> structure.
<a href="#">CD3D11_QUERY_DESC(D3D11_QUERY_DESC_values)</a> <a href="#">(D3D11_QUERY,UINT)</a>	Instantiates a new instance of a <b>CD3D11_QUERY_DESC</b> structure that is initialized with info for a query.

## Methods

The **CD3D11\_QUERY\_DESC** class has these methods.

[Expand table](#)

Method	Description
<a href="#">~CD3D11_QUERY_DESC()</a>	Destroys an instance of a <b>CD3D11_QUERY_DESC</b> structure.
<a href="#">CD3D11_QUERY_DESC(D3D11_QUERY_DESC)(const</a> <a href="#">&amp;D3D11_QUERY_DESC)</a>	Instantiates a new instance of a <b>CD3D11_QUERY_DESC</b> structure that is initialized with a <b>D3D11_QUERY_DESC</b> structure.

Method	Description
<a href="#">D3D11_QUERY_DESC()</a>	This operator returns the address of a <a href="#">D3D11_QUERY_DESC</a> structure that contains the data from the <a href="#">CD3D11_QUERY_DESC</a> instance.

## Remarks

Here is how D3D11.h defines [CD3D11\\_QUERY\\_DESC](#):

```
struct CD3D11_QUERY_DESC : public D3D11_QUERY_DESC
{
    CD3D11_QUERY_DESC()
    {}
    explicit CD3D11_QUERY_DESC( const D3D11_QUERY_DESC& o ) :
        D3D11_QUERY_DESC( o )
    {}
    explicit CD3D11_QUERY_DESC(
        D3D11_QUERY query,
        UINT miscFlags = 0 )
    {
        Query = query;
        MiscFlags = miscFlags;
    }
    ~CD3D11_QUERY_DESC() {}
    operator const D3D11_QUERY_DESC&() const { return *this; }
};
```

## Requirements

[Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h
Library	D3D11.lib

## See also

[D3D11\\_QUERY\\_DESC ↗](#)

[CD3D11 Helper Structures ↗](#)

# CD3D11\_QUERY\_DESC::CD3D11\_QUERY\_DESC function (d3d11.h)

Article 02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_QUERY\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_QUERY_DESC();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_QUERY\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_QUERY\_DESC::CD3D11\_QUERY\_DESC(const D3D11\_QUERY\_DESC&) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_QUERY\\_DESC](#) structure that is initialized with a [D3D11\\_QUERY\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_QUERY_DESC(
    [ref] const D3D11_QUERY_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_QUERY\\_DESC](#)

Address of the [D3D11\\_QUERY\\_DESC](#) structure that initializes the [CD3D11\\_QUERY\\_DESC](#) structure.

## Return value

None

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_QUERY\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_QUERY\_DESC::CD3D11\_QUERY\_DESC(D3D11\_QUERY,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_QUERY\\_DESC](#) structure that is initialized with info for a query.

## Syntax

C++

```
void CD3D11_QUERY_DESC(
    D3D11_QUERY query,
    UINT         miscFlags
);
```

## Parameters

`query`

Type: [D3D11\\_QUERY](#)

A [D3D11\\_QUERY](#)-typed value that specifies the type of query.

`miscFlags`

Type: [UINT](#)

A combination of [D3D11\\_QUERY\\_MISC\\_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies miscellaneous query behavior.

## Return value

None

## Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_QUERY\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_QUERY\_DESC::~CD3D11\_QUERY\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_QUERY\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_QUERY_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_QUERY\\_DESC](#)

# CD3D11\_COUNTER\_DESC class

Article 12/01/2017

Represents a counter and provides convenience methods for creating counters.

## Members

The CD3D11\_COUNTER\_DESC class inherits from [D3D11\\_COUNTER\\_DESC](#).

CD3D11\_COUNTER\_DESC also has these types of members:

- Constructors
- Methods

## Constructors

The CD3D11\_COUNTER\_DESC class has these constructors.

 Expand table

Constructor	Description
<a href="#">CD3D11_COUNTER_DESC()</a>	Instantiates a new instance of an uninitialized CD3D11_COUNTER_DESC structure.
<a href="#">CD3D11_COUNTER_DESC(D3D11_COUNTER_DESC_values)</a> <a href="#">(D3D11_COUNTER,UINT)</a>	Instantiates a new instance of a CD3D11_COUNTER_DESC structure that is initialized with info for a counter.

## Methods

The CD3D11\_COUNTER\_DESC class has these methods.

 Expand table

Method	Description
<a href="#">~CD3D11_COUNTER_DESC()</a>	Destroys an instance of a CD3D11_COUNTER_DESC structure.
<a href="#">CD3D11_COUNTER_DESC(D3D11_COUNTER_DESC)</a> <a href="#">(const &amp;D3D11_COUNTER_DESC)</a>	Instantiates a new instance of a CD3D11_COUNTER_DESC structure that is

Method	Description
	initialized with a <a href="#">D3D11_COUNTER_DESC</a> structure.
<a href="#">D3D11_COUNTER_DESC()</a>	This operator returns the address of a <a href="#">D3D11_COUNTER_DESC</a> structure that contains the data from the <a href="#">CD3D11_COUNTER_DESC</a> instance.

## Remarks

Here is how D3D11.h defines [CD3D11\\_COUNTER\\_DESC](#):

```
struct CD3D11_COUNTER_DESC : public D3D11_COUNTER_DESC
{
    CD3D11_COUNTER_DESC()
    {}
    explicit CD3D11_COUNTER_DESC( const D3D11_COUNTER_DESC& o ) :
        D3D11_COUNTER_DESC( o )
    {}
    explicit CD3D11_COUNTER_DESC(
        D3D11_COUNTER counter,
        UINT miscFlags = 0 )
    {
        Counter = counter;
        MiscFlags = miscFlags;
    }
    ~CD3D11_COUNTER_DESC() {}
    operator const D3D11_COUNTER_DESC&() const { return *this; }
};
```

## Requirements

[Expand table](#)

Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Header	D3D11.h

## See also

[D3D11\\_COUNTER\\_DESC](#)

[CD3D11 Helper Structures](#)

# CD3D11\_COUNTER\_DESC::CD3D11\_COUNTER\_DESC function (d3d11.h)

Article02/22/2024

Instantiates a new instance of an uninitialized [CD3D11\\_COUNTER\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_COUNTER_DESC();
```

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_COUNTER\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# CD3D11\_COUNTER\_DESC::CD3D11\_COUNTER\_DESC(const D3D11\_COUNTER\_DESC &) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_COUNTER\\_DESC](#) structure that is initialized with a [D3D11\\_COUNTER\\_DESC](#) structure.

## Syntax

C++

```
void CD3D11_COUNTER_DESC(
    [ref] const D3D11_COUNTER_DESC & o
);
```

## Parameters

[ref] o

Type: [const D3D11\\_COUNTER\\_DESC](#)

Address of the [D3D11\\_COUNTER\\_DESC](#) structure that initializes the [CD3D11\\_COUNTER\\_DESC](#) structure.

## Return value

None

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]

Requirement	Value
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_COUNTER\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_COUNTER\_DESC::CD3D11\_COUNTER\_DESC(D3D11\_COUNTER,UINT) function (d3d11.h)

Article02/22/2024

Instantiates a new instance of a [CD3D11\\_COUNTER\\_DESC](#) structure that is initialized with info for a counter.

## Syntax

C++

```
void CD3D11_COUNTER_DESC(
    D3D11_COUNTER counter,
    UINT          miscFlags
);
```

## Parameters

counter

Type: [D3D11\\_COUNTER](#)

A [D3D11\\_COUNTER](#)-typed value that specifies the type of counter.

miscFlags

Type: [UINT](#)

Reserved.

## Return value

None

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_COUNTER\\_DESC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CD3D11\_COUNTER\_DESC::~CD3D11\_COUNTER\_DESC function (d3d11.h)

Article02/22/2024

Destroys an instance of a [CD3D11\\_COUNTER\\_DESC](#) structure.

## Syntax

C++

```
void ~CD3D11_COUNTER_DESC();
```

## Return value

None

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps   UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps   UWP apps]
Target Platform	Windows
Header	d3d11.h
Library	D3D11.lib

## See also

[CD3D11\\_COUNTER\\_DESC](#)

# D3DCSX 11 Reference

Article • 11/04/2020

This section contains reference information about the D3DCSX utility library, which you can use with a compute shader.

## In this section

Topic	Description
D3DCSX 11 Interfaces	This section contains reference information about the COM interfaces provided by the D3DCSX utility library.
D3DCSX 11 Functions	This section contains information about the D3DCSX 11 functions.
D3DCSX 11 Structures	This section contains information about the D3DCSX 11 structures.
D3DCSX 11 Enumerations	This section contains information about D3DCSX 11 enumerations.

## Related topics

[Direct3D 11 Reference](#)

[Direct3D 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DCSX 11 Interfaces

Article • 11/04/2020

This section contains reference information about the COM interfaces provided by the D3DCSX utility library.

## In this section

Topic	Description
<a href="#">ID3DX11FFT</a>	Encapsulates forward and inverse FFTs.
<a href="#">ID3DX11Scan</a>	Scan context.
<a href="#">ID3DX11SegmentedScan</a>	Segmented scan context.

## Related topics

[D3DCSX 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11FFT interface (d3dcsx.h)

Article 07/22/2021

Encapsulates forward and inverse FFTs.

## Inheritance

The **ID3DX11FFT** interface inherits from the [IUnknown](#) interface. **ID3DX11FFT** also has these types of members:

## Methods

The **ID3DX11FFT** interface has these methods.

<a href="#">ID3DX11FFT::AttachBuffersAndPrecompute</a>
Attaches buffers to an FFT context and performs any required precomputations.
<a href="#">ID3DX11FFT::ForwardTransform</a>
Performs a forward FFT.
<a href="#">ID3DX11FFT::GetForwardScale</a>
Gets the scale for forward transforms.
<a href="#">ID3DX11FFT::GetInverseScale</a>
Get the scale for inverse transforms.
<a href="#">ID3DX11FFT::InverseTransform</a>
Performs an inverse FFT.
<a href="#">ID3DX11FFT::SetForwardScale</a>
Sets the scale used for forward transforms.
<a href="#">ID3DX11FFT::SetInverseScale</a>
Sets the scale used for inverse transforms.

# Requirements

Target Platform	Windows
Header	d3dcsx.h

## See also

[D3DCSX 11 Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX1FFT::AttachBuffersAndPrecompute method (d3dcsx.h)

Article 02/22/2024

Attaches buffers to an FFT context and performs any required precomputations.

## Syntax

C++

```
HRESULT AttachBuffersAndPrecompute(
    [in] UINT NumTempBuffers,
    [in] ID3D11UnorderedAccessView * const *ppTempBuffers,
    [in] UINT NumPrecomputeBuffers,
    [in] ID3D11UnorderedAccessView * const *ppPrecomputeBufferSizes
);
```

## Parameters

[in] NumTempBuffers

Type: [UINT](#)

Number of buffers in *ppTempBuffers*.

[in] ppTempBuffers

Type: [ID3D11UnorderedAccessView\\*](#)

A pointer to an array of [ID3D11UnorderedAccessView](#) pointers for the temporary buffers to attach. The FFT object might use these temporary buffers for its algorithm.

[in] NumPrecomputeBuffers

Type: [UINT](#)

Number of buffers in *ppPrecomputeBuffers*.

[in] ppPrecomputeBufferSizes

Type: [ID3D11UnorderedAccessView\\*](#)

A pointer to an array of [ID3D11UnorderedAccessView](#) pointers for the precompute buffers to attach. The FFT object might store precomputed data in these buffers.

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

The [D3DX11\\_FFT\\_BUFFER\\_INFO](#) structure is initialized by a call to one of the create-FFT functions (for example, [D3DX11CreateFFT](#)). For more create-FFT functions, see [D3DCSX 11 Functions](#).

Use the info in [D3DX11\\_FFT\\_BUFFER\\_INFO](#) to allocate raw buffers of the specified (or larger) sizes and then call the [AttachBuffersAndPrecompute](#) to register the buffers with the FFT object.

Although you can share temporary buffers between multiple device contexts, we recommend not to concurrently execute multiple FFT objects that share temporary buffers.

Some FFT algorithms benefit from precomputing sin and cos. The FFT object might store precomputed data in the user-supplied precompute buffers.

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsv.h
Library	D3dcsv.lib

## See also

[ID3DX11FFT](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DX1FFT::ForwardTransform method (d3dcsx.h)

Article 02/22/2024

Performs a forward FFT.

## Syntax

C++

```
HRESULT ForwardTransform(
    [in]      const ID3D11UnorderedAccessView *pInputBuffer,
    [in, out] ID3D11UnorderedAccessView       **ppOutputBuffer
);
```

## Parameters

[in] pInputBuffer

Type: [const ID3D11UnorderedAccessView\\*](#)

Pointer to [ID3D11UnorderedAccessView](#) onto the input buffer.

[in, out] ppOutputBuffer

Type: [ID3D11UnorderedAccessView\\*\\*](#)

Pointer to a [ID3D11UnorderedAccessView](#) pointer. If *\*ppOutputBuffer* is **NULL**, the computation will switch between temp buffers; in addition, the last buffer written to is stored at *\*ppOutputBuffer*. Otherwise, *\*ppOutputBuffer* is used as the output buffer (which might incur an extra copy).

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

**ForwardTransform** can be called after buffers have been attached to the context using [ID3DX11FFT::AttachBuffersAndPrecompute](#). The combination of *pInputBuffer* and *\*ppOutputBuffer* can be one of the temp buffers.

The format of complex data is interleaved components (for example, (Real0, Imag0), (Real1, Imag1) ... , and so on). Data is stored in row major order.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11FFT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DX11FFT::GetForwardScale method (d3dcsx.h)

Article 02/22/2024

Gets the scale for forward transforms.

## Syntax

C++

```
FLOAT GetForwardScale();
```

## Return value

Type: [FLOAT](#)

Scale for forward transforms.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11FFT](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# ID3DX11FFT::GetInverseScale method (d3dcsx.h)

Article 02/22/2024

Get the scale for inverse transforms.

## Syntax

C++

```
FLOAT GetInverseScale();
```

## Return value

Type: [FLOAT](#)

Scale for inverse transforms.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11FFT](#)

---

## Feedback

Was this page helpful?

 Yes

 No



# ID3DX1FFT::InverseTransform method (d3dcsx.h)

Article 02/22/2024

Performs an inverse FFT.

## Syntax

C++

```
HRESULT InverseTransform(
    [in]      const ID3D11UnorderedAccessView *pInputBuffer,
    [in, out] ID3D11UnorderedAccessView       **ppOutputBuffer
);
```

## Parameters

[in] pInputBuffer

Type: [const ID3D11UnorderedAccessView\\*](#)

Pointer to [ID3D11UnorderedAccessView](#) onto the input buffer.

[in, out] ppOutputBuffer

Type: [ID3D11UnorderedAccessView\\*\\*](#)

Pointer to a [ID3D11UnorderedAccessView](#) pointer. If *\*ppOutput* is **NULL**, then the computation will switch between temp buffers; in addition, the last buffer written to is stored at *\*ppOutput*. Otherwise, *\*ppOutput* is used as the output buffer (which might incur an extra copy).

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Requirements

[ ] Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11FFT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3DX11FFT::SetForwardScale method (d3dcsx.h)

Article 02/22/2024

Sets the scale used for forward transforms.

## Syntax

C++

```
HRESULT SetForwardScale(  
    [in] FLOAT ForwardScale  
);
```

## Parameters

[in] ForwardScale

Type: [FLOAT](#)

The scale to use for forward transforms. Setting *ForwardScale* to 0 causes the default values of 1 to be used.

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

`SetForwardScale` sets the scale used by [ID3DX11FFT::ForwardTransform](#).

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11FFT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DX11FFT::SetInverseScale method (d3dcsx.h)

Article 02/22/2024

Sets the scale used for inverse transforms.

## Syntax

C++

```
HRESULT SetInverseScale(  
    [in] FLOAT InverseScale  
);
```

## Parameters

[in] InverseScale

Type: [FLOAT](#)

Scale used for inverse transforms. Setting *InverseScale* to 0 causes the default value of  $1/N$  to be used, where N is the product of the transformed dimension lengths.

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

`SetInverseScale` sets the scale used by [ID3DX11FFT::InverseTransform](#).

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11FFT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DX11Scan interface (d3dcsx.h)

Article 02/22/2024

Scan context.

## Inheritance

The **ID3DX11Scan** interface inherits from the [IUnknown](#) interface. **ID3DX11Scan** also has these types of members:

## Methods

The **ID3DX11Scan** interface has these methods.

[+] [Expand table](#)

<b>ID3DX11Scan::Multiscan</b>
Performs a multiscan of a sequence.
<b>ID3DX11Scan::Scan</b>
Performs an unsegmented scan of a sequence.
<b>ID3DX11Scan::SetScanDirection</b>
Sets which direction to perform scans in. ( <code>ID3DX11Scan.SetScanDirection</code> )

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DX11Scan::Multiscan method (d3dcsx.h)

Article 10/13/2021

Performs a multiscan of a sequence.

## Syntax

C++

```
HRESULT Multiscan(
    [in] D3DX11_SCAN_DATA_TYPE      ElementType,
    [in] D3DX11_SCAN_OPCODE        OpCode,
    [in] UINT                      ElementScanSize,
    [in] UINT                      ElementScanPitch,
    [in] UINT                      ScanCount,
    [in] ID3D11UnorderedAccessView *pSrc,
    [in] ID3D11UnorderedAccessView *pDst
);
```

## Parameters

[in] ElementType

Type: [D3DX11\\_SCAN\\_DATA\\_TYPE](#)

The type of element in the sequence. See [D3DX11\\_SCAN\\_DATA\\_TYPE](#) for more information.

[in] OpCode

Type: [D3DX11\\_SCAN\\_OPCODE](#)

The binary operation to perform. See [D3DX11\\_SCAN\\_OPCODE](#) for more information.

[in] ElementScanSize

Type: [UINT](#)

Size of scan in elements.

[in] ElementScanPitch

Type: [UINT](#)

Pitch of the next scan in elements.

[in] ScanCount

Type: [UINT](#)

Number of scans in the multiscan.

[in] pSrc

Type: [ID3D11UnorderedAccessView\\*](#)

Input sequence on the device. Set *pSrc* and *pDst* to the same value for in-place scans.

[in] pDst

Type: [ID3D11UnorderedAccessView\\*](#)

Output sequence on the device.

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

You must point the parameters *pSrc* and *pDst* to typed buffers (and not to raw or structured buffers). For information about buffer types, see [Types of Resources](#). The format of these typed buffers must be [DXGI\\_FORMAT\\_R32\\_FLOAT](#), [DXGI\\_FORMAT\\_R32\\_UINT](#), or [DXGI\\_FORMAT\\_R32\\_INT](#). In addition, the format of these typed buffers must match the scan data type that you specify in the *ElementType* parameter. For example, if the scan data type is [D3DX11\\_SCAN\\_DATA\\_TYPE\\_UINT](#), the buffer formats must be [DXGI\\_FORMAT\\_R32\\_UINT](#).

## Requirements

Target Platform	Windows
-----------------	---------

Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11Scan](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DX11Scan::Scan method (d3dcsx.h)

Article02/22/2024

Performs an unsegmented scan of a sequence.

## Syntax

C++

```
HRESULT Scan(
    [in] D3DX11_SCAN_DATA_TYPE      ElementType,
    [in] D3DX11_SCAN_OPCODE        OpCode,
    [in] UINT                      ElementScanSize,
    [in] ID3D11UnorderedAccessView *pSrc,
    [in] ID3D11UnorderedAccessView *pDst
);
```

## Parameters

[in] ElementType

Type: [D3DX11\\_SCAN\\_DATA\\_TYPE](#)

The type of element in the sequence. See [D3DX11\\_SCAN\\_DATA\\_TYPE](#) for more information.

[in] OpCode

Type: [D3DX11\\_SCAN\\_OPCODE](#)

The binary operation to perform. See [D3DX11\\_SCAN\\_OPCODE](#) for more information.

[in] ElementScanSize

Type: [UINT](#)

Size of scan in elements.

[in] pSrc

Type: [ID3D11UnorderedAccessView\\*](#)

Input sequence on the device. Set *pSrc* and *pDst* to the same value for in-place scans.

[in] pDst

Type: [ID3D11UnorderedAccessView\\*](#)

Output sequence on the device.

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

You must point the parameters *pSrc* and *pDst* to typed buffers (and not to raw or structured buffers). For information about buffer types, see [Types of Resources](#). The format of these typed buffers must be [DXGI\\_FORMAT\\_R32\\_FLOAT](#), [DXGI\\_FORMAT\\_R32\\_UINT](#), or [DXGI\\_FORMAT\\_R32\\_INT](#). In addition, the format of these typed buffers must match the scan data type that you specify in the *ElementType* parameter. For example, if the scan data type is [D3DX11\\_SCAN\\_DATA\\_TYPE\\_UINT](#), the buffer formats must be [DXGI\\_FORMAT\\_R32\\_UINT](#).

## Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11Scan](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# ID3DX11Scan::SetScanDirection method (d3dcsx.h)

Article 02/22/2024

Sets which direction to perform scans in.

## Syntax

C++

```
HRESULT SetScanDirection(  
    [in] D3DX11_SCAN_DIRECTION Direction  
);
```

## Parameters

[in] `Direction`

Type: [D3DX11\\_SCAN\\_DIRECTION](#)

Direction to perform scans in. See [D3DX11\\_SCAN\\_DIRECTION](#).

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

`SetScanDirection` sets the direction `ID3DX11Scan::Scan` and `ID3DX11Scan::Multiscan` will performed scans in.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11Scan](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DX11SegmentedScan interface (d3dcsx.h)

Article 02/22/2024

Segmented scan context.

## Inheritance

The **ID3DX11SegmentedScan** interface inherits from the [IUnknown](#) interface.

**ID3DX11SegmentedScan** also has these types of members:

## Methods

The **ID3DX11SegmentedScan** interface has these methods.

[+] Expand table

<a href="#">ID3DX11SegmentedScan::SegScan</a>
Performs a segmented scan of a sequence.

<a href="#">ID3DX11SegmentedScan::SetScanDirection</a>
Sets which direction to perform scans in. ( <code>ID3DX11SegmentedScan.SetScanDirection</code> )

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h

## See also

[D3DCSX 11 Interfaces](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DX11SegmentedScan::SegScan method (d3dcsx.h)

Article 02/22/2024

Performs a segmented scan of a sequence.

## Syntax

C++

```
HRESULT SegScan(
    [in] D3DX11_SCAN_DATA_TYPE      ElementType,
    [in] D3DX11_SCAN_OPCODE        OpCode,
    [in] UINT                      ElementScanSize,
    [in] ID3D11UnorderedAccessView *pSrc,
    [in] ID3D11UnorderedAccessView *pSrcElementFlags,
    [in] ID3D11UnorderedAccessView *pDst
);
```

## Parameters

[in] ElementType

Type: [D3DX11\\_SCAN\\_DATA\\_TYPE](#)

The type of element in the sequence. See [D3DX11\\_SCAN\\_DATA\\_TYPE](#) for more information.

[in] OpCode

Type: [D3DX11\\_SCAN\\_OPCODE](#)

The binary operation to perform. See [D3DX11\\_SCAN\\_OPCODE](#) for more information.

[in] ElementScanSize

Type: [UINT](#)

Size of scan in elements.

[in] pSrc

Type: [ID3D11UnorderedAccessView\\*](#)

Input sequence on the device. Set *pSrc* and *pDst* to the same value for in-place scans.

[in] *pSrcElementFlags*

Type: [ID3D11UnorderedAccessView\\*](#)

Compact array of bits with one bit per element of *pSrc*. A set value indicates the start of a new segment.

[in] *pDst*

Type: [ID3D11UnorderedAccessView\\*](#)

Output sequence on the device.

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

You must point the parameters *pSrc* and *pDst* to typed buffers (and not to raw or structured buffers). For information about buffer types, see [Types of Resources](#). The format of these typed buffers must be [DXGI\\_FORMAT\\_R32\\_FLOAT](#), [DXGI\\_FORMAT\\_R32\\_UINT](#), or [DXGI\\_FORMAT\\_R32\\_INT](#). In addition, the format of these typed buffers must match the scan data type that you specify in the *ElementType* parameter. For example, if the scan data type is [D3DX11\\_SCAN\\_DATA\\_TYPE\\_UINT](#), the buffer formats must be [DXGI\\_FORMAT\\_R32\\_UINT](#).

The format of the resource view to which the *pSrcElementFlags* parameter points must be [DXGI\\_FORMAT\\_R32\\_UINT](#).

## Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h

Requirement	Value
Library	D3dcsx.lib

## See also

[ID3DX11SegmentedScan](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# ID3DX11SegmentedScan::SetScanDirection method (d3dcsx.h)

Article 02/22/2024

Sets which direction to perform scans in.

## Syntax

C++

```
HRESULT SetScanDirection(  
    [in] D3DX11_SCAN_DIRECTION Direction  
);
```

## Parameters

[in] `Direction`

Type: [D3DX11\\_SCAN\\_DIRECTION](#)

Direction to perform scans in. See [D3DX11\\_SCAN\\_DIRECTION](#).

## Return value

Type: [HRESULT](#)

Returns one of the return codes described in the topic [Direct3D 11 Return Codes](#).

## Remarks

`SetScanDirection` sets the direction `ID3DX11SegmentedScan::SegScan` will performed scans in.

## Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[ID3DX11SegmentedScan](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DCSX 11 Functions

Article • 11/04/2020

This section contains information about the D3DCSX 11 functions.

## In this section

Topic	Description
<a href="#">D3DX11CreateScan</a>	Creates a scan context.
<a href="#">D3DX11CreateSegmentedScan</a>	Creates a segmented scan context.
<a href="#">D3DX11CreateFFT</a>	Creates an <a href="#">ID3DX11FFT</a> COM interface object.
<a href="#">D3DX11CreateFFT1DComplex</a>	Creates an <a href="#">ID3DX11FFT</a> COM interface object.
<a href="#">D3DX11CreateFFT1DReal</a>	Creates an <a href="#">ID3DX11FFT</a> COM interface object.
<a href="#">D3DX11CreateFFT2DComplex</a>	Creates an <a href="#">ID3DX11FFT</a> COM interface object.
<a href="#">D3DX11CreateFFT2DReal</a>	Creates an <a href="#">ID3DX11FFT</a> COM interface object.
<a href="#">D3DX11CreateFFT3DComplex</a>	Creates an <a href="#">ID3DX11FFT</a> COM interface object.
<a href="#">D3DX11CreateFFT3DReal</a>	Creates an <a href="#">ID3DX11FFT</a> COM interface object.

## Related topics

[D3DCSX 11 Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateScan function (d3dcsx.h)

Article 02/22/2024

Creates a scan context.

## Syntax

C++

```
HRESULT D3DX11CreateScan(
    [in] ID3D11DeviceContext *pDeviceContext,
    [in]    UINT             MaxElementScanSize,
    [in]    UINT             MaxScanCount,
    [out] ID3DX11Scan     **ppScan
);
```

## Parameters

[in] pDeviceContext

Type: [ID3D11DeviceContext\\*](#)

The [ID3D11DeviceContext](#) the scan is associated with.

MaxElementScanSize

Type: [UINT](#)

Maximum single scan size, in elements (FLOAT, UINT, or INT).

MaxScanCount

Type: [UINT](#)

Maximum number of scans in multiscan.

[out] ppScan

Type: [ID3DX11Scan\\*\\*](#)

Pointer to a [ID3DX11Scan](#) Interface pointer that will be set to the created interface object.

# Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[D3DCSX 11 Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# D3DX11CreateSegmentedScan function (d3dcsx.h)

Article 02/22/2024

Creates a segmented scan context.

## Syntax

C++

```
HRESULT D3DX11CreateSegmentedScan(
    [in] ID3D11DeviceContext  *pDeviceContext,
    [in]          UINT        MaxElementScanSize,
    [out] ID3DX11SegmentedScan **ppScan
);
```

## Parameters

[in] pDeviceContext

Type: [ID3D11DeviceContext\\*](#)

Pointer to an [ID3D11DeviceContext](#) interface.

MaxElementScanSize

Type: [UINT](#)

Maximum single scan size, in elements (FLOAT, UINT, or INT).

[out] ppScan

Type: [ID3DX11SegmentedScan\\*\\*](#)

Pointer to a [ID3DX11SegmentedScan Interface](#) pointer that will be set to the created interface object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

# Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[D3DCSX 11 Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# D3DX11CreateFFT function (d3dcsx.h)

Article 02/22/2024

Creates an [ID3DX11FFT](#) COM interface object.

## Syntax

C++

```
HRESULT D3DX11CreateFFT(
    ID3D11DeviceContext     *pDeviceContext,
    [in]  const D3DX11_FFT_DESC  *pDesc,
    UINT                      Flags,
    [out] D3DX11_FFT_BUFFER_INFO *pBufferInfo,
    [out] ID3DX11FFT          **ppFFT
);
```

## Parameters

`pDeviceContext`

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#) interface to use for the FFT.

`[in] pDesc`

Type: [const D3DX11\\_FFT\\_DESC\\*](#)

A pointer to a [D3DX11\\_FFT\\_DESC](#) structure that describes the shape of the FFT data as well as the scaling factors that should be used for forward and inverse transforms.

`Flags`

Type: [UINT](#)

Flags that affect the behavior of the FFT, can be 0 or a combination of flags from [D3DX11\\_FFT\\_CREATE\\_FLAG](#).

`[out] pBufferInfo`

Type: [D3DX11\\_FFT\\_BUFFER\\_INFO\\*](#)

A pointer to a [D3DX11\\_FFT\\_BUFFER\\_INFO](#) structure that receives the buffer requirements to execute the FFT algorithms. Use this info to allocate raw buffers of the specified (or larger) sizes and then call the [ID3DX11FFT::AttachBuffersAndPrecompute](#) method to register the buffers with the FFT object.

[out] ppFFT

Type: [ID3DX11FFT\\*\\*](#)

A pointer to a variable that receives a pointer to the [ID3DX11FFT](#) interface for the created FFT object.

## Return value

Type: [HRESULT](#)

One of the [Direct3D 11 Return Codes](#).

## Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[D3DCSX 11 Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# D3DX11CreateFFT1DComplex function (d3dcsx.h)

Article 02/22/2024

Creates an [ID3DX11FFT](#) COM interface object.

## Syntax

C++

```
HRESULT D3DX11CreateFFT1DComplex(
    ID3D11DeviceContext      *pDeviceContext,
    UINT                      X,
    UINT                      Flags,
    [out] D3DX11_FFT_BUFFER_INFO *pBufferInfo,
    [out] ID3DX11FFT          **ppFFT
);
```

## Parameters

pDeviceContext

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#) interface to use for the FFT.

X

Type: [UINT](#)

Length of the first dimension of the FFT.

Flags

Type: [UINT](#)

Flags that affect the behavior of the FFT, can be 0 or a combination of flags from [D3DX11\\_FFT\\_CREATE\\_FLAG](#).

[out] pBufferInfo

Type: [D3DX11\\_FFT\\_BUFFER\\_INFO\\*](#)

A pointer to a [D3DX11\\_FFT\\_BUFFER\\_INFO](#) structure that receives the buffer requirements to execute the FFT algorithms. Use this info to allocate raw buffers of the specified (or larger) sizes and then call the [ID3DX11FFT::AttachBuffersAndPrecompute](#) method to register the buffers with the FFT object.

[out] ppFFT

Type: [ID3DX11FFT\\*\\*](#)

A pointer to a variable that receives a pointer to the [ID3DX11FFT](#) interface for the created FFT object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[D3DCSX 11 Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11CreateFFT1DReal function (d3dcsx.h)

Article 02/22/2024

Creates an [ID3DX11FFT](#) COM interface object.

## Syntax

C++

```
HRESULT D3DX11CreateFFT1DReal(
    ID3D11DeviceContext      *pDeviceContext,
    UINT                      X,
    UINT                      Flags,
    [out] D3DX11_FFT_BUFFER_INFO *pBufferInfo,
    [out] ID3DX11FFT          **ppFFT
);
```

## Parameters

pDeviceContext

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#) interface to use for the FFT.

X

Type: [UINT](#)

Length of the first dimension of the FFT.

Flags

Type: [UINT](#)

Flags that affect the behavior of the FFT, can be 0 or a combination of flags from [D3DX11\\_FFT\\_CREATE\\_FLAG](#).

[out] pBufferInfo

Type: [D3DX11\\_FFT\\_BUFFER\\_INFO\\*](#)

A pointer to a [D3DX11\\_FFT\\_BUFFER\\_INFO](#) structure that receives the buffer requirements to execute the FFT algorithms. Use this info to allocate raw buffers of the specified (or larger) sizes and then call the [ID3DX11FFT::AttachBuffersAndPrecompute](#) method to register the buffers with the FFT object.

[out] ppFFT

Type: [ID3DX11FFT\\*\\*](#)

A pointer to a variable that receives a pointer to the [ID3DX11FFT](#) interface for the created FFT object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[D3DCSX 11 Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11CreateFFT2DComplex function (d3dcsx.h)

Article 02/22/2024

Creates an [ID3DX11FFT](#) COM interface object.

## Syntax

C++

```
HRESULT D3DX11CreateFFT2DComplex(
    ID3D11DeviceContext      *pDeviceContext,
    UINT                      X,
    UINT                      Y,
    UINT                      Flags,
    [out] D3DX11_FFT_BUFFER_INFO *pBufferInfo,
    [out] ID3DX11FFT          **ppFFT
);
```

## Parameters

pDeviceContext

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#) interface to use for the FFT.

X

Type: [UINT](#)

Length of the first dimension of the FFT.

Y

Type: [UINT](#)

Length of the second dimension of the FFT.

Flags

Type: [UINT](#)

Flags that affect the behavior of the FFT, can be 0 or a combination of flags from [D3DX11\\_FFT\\_CREATE\\_FLAG](#).

[out] pBufferInfo

Type: [D3DX11\\_FFT\\_BUFFER\\_INFO](#)\*

A pointer to a [D3DX11\\_FFT\\_BUFFER\\_INFO](#) structure that receives the buffer requirements to execute the FFT algorithms. Use this info to allocate raw buffers of the specified (or larger) sizes and then call the [ID3DX11FFT::AttachBuffersAndPrecompute](#) method to register the buffers with the FFT object.

[out] ppFFT

Type: [ID3DX11FFT](#)\*\*

A pointer to a variable that receives a pointer to the [ID3DX11FFT](#) interface for the created FFT object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[D3DCSX 11 Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11CreateFFT2DReal function (d3dcsx.h)

Article 02/22/2024

Creates an [ID3DX11FFT](#) COM interface object.

## Syntax

C++

```
HRESULT D3DX11CreateFFT2DReal(
    ID3D11DeviceContext      *pDeviceContext,
    UINT                      X,
    UINT                      Y,
    UINT                      Flags,
    [out] D3DX11_FFT_BUFFER_INFO *pBufferInfo,
    [out] ID3DX11FFT          **ppFFT
);
```

## Parameters

pDeviceContext

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#) interface to use for the FFT.

X

Type: [UINT](#)

Length of the first dimension of the FFT.

Y

Type: [UINT](#)

Length of the second dimension of the FFT.

Flags

Type: [UINT](#)

Flags that affect the behavior of the FFT, can be 0 or a combination of flags from [D3DX11\\_FFT\\_CREATE\\_FLAG](#).

[out] pBufferInfo

Type: [D3DX11\\_FFT\\_BUFFER\\_INFO](#)\*

A pointer to a [D3DX11\\_FFT\\_BUFFER\\_INFO](#) structure that receives the buffer requirements to execute the FFT algorithms. Use this info to allocate raw buffers of the specified (or larger) sizes and then call the [ID3DX11FFT::AttachBuffersAndPrecompute](#) method to register the buffers with the FFT object.

[out] ppFFT

Type: [ID3DX11FFT](#)\*\*

A pointer to a variable that receives a pointer to the [ID3DX11FFT](#) interface for the created FFT object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

  Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

[D3DCSX 11 Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

# D3DX11CreateFFT3DComplex function (d3dcsx.h)

Article 02/22/2024

Creates an [ID3DX11FFT](#) COM interface object.

## Syntax

C++

```
HRESULT D3DX11CreateFFT3DComplex(
    ID3D11DeviceContext      *pDeviceContext,
    UINT                      X,
    UINT                      Y,
    UINT                      Z,
    UINT                      Flags,
    [out] D3DX11_FFT_BUFFER_INFO *pBufferInfo,
    [out] ID3DX11FFT           **ppFFT
);
```

## Parameters

`pDeviceContext`

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#) interface to use for the FFT.

`X`

Type: [UINT](#)

Length of the first dimension of the FFT.

`Y`

Type: [UINT](#)

Length of the second dimension of the FFT.

`Z`

Type: [UINT](#)

Length of the third dimension of the FFT.

#### Flags

Type: [UINT](#)

Flags that affect the behavior of the FFT, can be 0 or a combination of flags from [D3DX11\\_FFT\\_CREATE\\_FLAG](#).

#### [out] pBufferInfo

Type: [D3DX11\\_FFT\\_BUFFER\\_INFO\\*](#)

A pointer to a [D3DX11\\_FFT\\_BUFFER\\_INFO](#) structure that receives the buffer requirements to execute the FFT algorithms. Use this info to allocate raw buffers of the specified (or larger) sizes and then call the [ID3DX11FFT::AttachBuffersAndPrecompute](#) method to register the buffers with the FFT object.

#### [out] ppFFT

Type: [ID3DX11FFT\\*\\*](#)

A pointer to a variable that receives a pointer to the [ID3DX11FFT](#) interface for the created FFT object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11CreateFFT3DReal function (d3dcsx.h)

Article 02/22/2024

Creates an [ID3DX11FFT](#) COM interface object.

## Syntax

C++

```
HRESULT D3DX11CreateFFT3DReal(
    ID3D11DeviceContext      *pDeviceContext,
    UINT                      X,
    UINT                      Y,
    UINT                      Z,
    UINT                      Flags,
    [out] D3DX11_FFT_BUFFER_INFO *pBufferInfo,
    [out] ID3DX11FFT          **ppFFT
);
```

## Parameters

pDeviceContext

Type: [ID3D11DeviceContext\\*](#)

A pointer to the [ID3D11DeviceContext](#) interface to use for the FFT.

X

Type: [UINT](#)

Length of the first dimension of the FFT.

Y

Type: [UINT](#)

Length of the second dimension of the FFT.

Z

Type: [UINT](#)

Length of the third dimension of the FFT.

#### Flags

Type: [UINT](#)

Flags that affect the behavior of the FFT, can be 0 or a combination of flags from [D3DX11\\_FFT\\_CREATE\\_FLAG](#).

#### [out] pBufferInfo

Type: [D3DX11\\_FFT\\_BUFFER\\_INFO\\*](#)

A pointer to a [D3DX11\\_FFT\\_BUFFER\\_INFO](#) structure that receives the buffer requirements to execute the FFT algorithms. Use this info to allocate raw buffers of the specified (or larger) sizes and then call the [ID3DX11FFT::AttachBuffersAndPrecompute](#) method to register the buffers with the FFT object.

#### [out] ppFFT

Type: [ID3DX11FFT\\*\\*](#)

A pointer to a variable that receives a pointer to the [ID3DX11FFT](#) interface for the created FFT object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3dcsx.h
Library	D3dcsx.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DCSX 11 Structures

Article • 11/04/2020

This section contains information about the D3DCSX 11 structures.

## In this section

Topic	Description
<a href="#">D3DX11_FFT_BUFFER_INFO</a>	Describes buffer requirements for an FFT.
<a href="#">D3DX11_FFT_DESC</a>	Describes an FFT.

## Related topics

[D3DCSX 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_FFT\_BUFFER\_INFO structure (d3dcsx.h)

Article02/22/2024

Describes buffer requirements for an FFT.

## Syntax

C++

```
typedef struct D3DX11_FFT_BUFFER_INFO {
    UINT NumTempBufferSizes;
    UINT TempBufferFloatSizes[D3DX11_FFT_MAX_TEMP_BUFFERS];
    UINT NumPrecomputeBufferSizes;
    UINT PrecomputeBufferFloatSizes[D3DX11_FFT_MAX_PRECOMPUTE_BUFFERS];
} D3DX11_FFT_BUFFER_INFO;
```

## Members

NumTempBufferSizes

Type: **UINT**

Number of temporary buffers needed. Allowed range is 0 to D3DX11\_FFT\_MAX\_TEMP\_BUFFERS.

TempBufferFloatSizes[D3DX11\_FFT\_MAX\_TEMP\_BUFFERS]

Type: **UINT[D3DX11\_FFT\_MAX\_TEMP\_BUFFERS]**

Minimum sizes (in FLOATs) of temporary buffers.

NumPrecomputeBufferSizes

Type: **UINT**

Number of precompute buffers required.

Allowed range is 0 to D3DX11\_FFT\_MAX\_PRECOMPUTE\_BUFFERS.

PrecomputeBufferFloatSizes[D3DX11\_FFT\_MAX\_PRECOMPUTE\_BUFFERS]

Type: **UINT[D3DX11\_FFT\_MAX\_PRECOMPUTE\_BUFFERS]**

Minimum sizes (in FLOATs) for precompute buffers.

## Remarks

The `D3DX11_FFT_BUFFER_INFO` structure is initialized by a call to one of the create-FFT functions (for example, [D3DX11CreateFFT](#)). For more create-FFT functions, see [D3DCSX 11 Functions](#).

Use the info in `D3DX11_FFT_BUFFER_INFO` to allocate raw buffers of the specified (or larger) sizes and then call the [`ID3DX11FFT::AttachBuffersAndPrecompute`](#) method to register the buffers with the FFT object.

Some FFT algorithms benefit from precomputing sin and cos. The FFT object might store precomputed data in the user-supplied buffers.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Header	d3dcsx.h

## See also

[D3DCSX 11 Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11\_FFT\_DESC structure (d3dcsx.h)

Article 02/22/2024

Describes an FFT.

## Syntax

C++

```
typedef struct D3DX11_FFT_DESC {
    UINT             NumDimensions;
    UINT             ElementLengths[D3DX11_FFT_MAX_DIMENSIONS];
    UINT             DimensionMask;
    D3DX11_FFT_DATA_TYPE Type;
} D3DX11_FFT_DESC;
```

## Members

`NumDimensions`

Type: **UINT**

Number of dimension in the FFT.

`ElementLengths[D3DX11_FFT_MAX_DIMENSIONS]`

Type: **UINT[D3DX11\_FFT\_MAX\_DIMENSIONS]**

Length of each dimension in the FFT.

`DimensionMask`

Type: **UINT**

Combination of **D3DX11\_FFT\_DIM\_MASK** flags indicating the dimensions to transform.

`Type`

Type: **D3DX11\_FFT\_DATA\_TYPE**

**D3DX11\_FFT\_DATA\_TYPE** flag indicating the type of data being transformed.

## Requirements

Expand table

Requirement	Value
Header	d3dcsx.h

## See also

[D3DCSX 11 Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# D3DCSX 11 Enumerations

Article • 11/04/2020

This section contains information about D3DCSX 11 enumerations.

## In this section

Topic	Description
<a href="#">D3DX11_FFT_CREATE_FLAG</a>	FFT creation flags.
<a href="#">D3DX11_FFT_DATA_TYPE</a>	FFT data types.
<a href="#">D3DX11_FFT_DIM_MASK</a>	Number of dimensions for FFT data.
<a href="#">D3DX11_SCAN_DATA_TYPE</a>	Type for scan data.
<a href="#">D3DX11_SCAN_DIRECTION</a>	Direction to perform scan in.
<a href="#">D3DX11_SCAN_OPCODE</a>	Scan opcodes.

## Related topics

[D3DCSX 11 Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_FFT\_CREATE\_FLAG enumeration (d3dcsx.h)

Article 02/22/2024

FFT creation flags.

## Syntax

C++

```
typedef enum D3DX11_FFT_CREATE_FLAG {
    D3DX11_FFT_CREATE_FLAG_NO_PRECOMPUTE_BUFFERS = 0x01L
} ;
```

## Constants

[+] Expand table

D3DX11\_FFT\_CREATE\_FLAG\_NO\_PRECOMPUTE\_BUFFERS

Value: *0x01L*

Do not AddRef or Release temp and precompute buffers, caller is responsible for holding references to these buffers.

## Requirements

[+] Expand table

Requirement	Value
Header	d3dcsx.h

## See also

[D3DCSX 11 Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11\_FFT\_DATA\_TYPE enumeration (d3dcsx.h)

Article 02/22/2024

FFT data types.

## Syntax

C++

```
typedef enum D3DX11_FFT_DATA_TYPE {
    D3DX11_FFT_DATA_TYPE_REAL,
    D3DX11_FFT_DATA_TYPE_COMPLEX
} ;
```

## Constants

[+] Expand table

<code>D3DX11_FFT_DATA_TYPE_REAL</code>	Real numbers.
<code>D3DX11_FFT_DATA_TYPE_COMPLEX</code>	Complex numbers.

## Requirements

[+] Expand table

Requirement	Value
Header	d3dcsx.h

## See also

[D3DCSX 11 Enumerations](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11\_FFT\_DIM\_MASK enumeration (d3dcsx.h)

Article 02/22/2024

Number of dimensions for FFT data.

## Syntax

C++

```
typedef enum D3DX11_FFT_DIM_MASK {
    D3DX11_FFT_DIM_MASK_1D = 0x1,
    D3DX11_FFT_DIM_MASK_2D = 0x3,
    D3DX11_FFT_DIM_MASK_3D = 0x7
};
```

## Constants

[+] Expand table

<b>D3DX11_FFT_DIM_MASK_1D</b> Value: 0x1 One dimension.
<b>D3DX11_FFT_DIM_MASK_2D</b> Value: 0x3 Two dimensions.
<b>D3DX11_FFT_DIM_MASK_3D</b> Value: 0x7 Three dimensions.

## Requirements

[+] Expand table

Requirement	Value
Header	d3dcsx.h

## See also

[D3DCSX 11 Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11\_SCAN\_DATA\_TYPE enumeration (d3dcsx.h)

Article 02/22/2024

Type for scan data.

## Syntax

C++

```
typedef enum D3DX11_SCAN_DATA_TYPE {
    D3DX11_SCAN_DATA_TYPE_FLOAT = 1,
    D3DX11_SCAN_DATA_TYPE_INT,
    D3DX11_SCAN_DATA_TYPE_UINT
};
```

## Constants

[+] Expand table

D3DX11\_SCAN\_DATA\_TYPE\_FLOAT

Value: 1

FLOAT data.

D3DX11\_SCAN\_DATA\_TYPE\_INT

INT data.

D3DX11\_SCAN\_DATA\_TYPE\_UINT

UINT data.

## Requirements

[+] Expand table

Requirement	Value
Header	d3dcsx.h

## See also

[D3DCSX 11 Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11\_SCAN\_DIRECTION enumeration (d3dcsx.h)

Article 02/22/2024

Direction to perform scan in.

## Syntax

C++

```
typedef enum D3DX11_SCAN_DIRECTION {
    D3DX11_SCAN_DIRECTION_FORWARD = 1,
    D3DX11_SCAN_DIRECTION_BACKWARD
} ;
```

## Constants

[+] Expand table

	<code>D3DX11_SCAN_DIRECTION_FORWARD</code>
Value:	1
Scan forward.	
	<code>D3DX11_SCAN_DIRECTION_BACKWARD</code>
Scan backward.	

## Requirements

[+] Expand table

Requirement	Value
Header	d3dcsx.h

## See also

[D3DCSX 11 Enumerations](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# D3DX11\_SCAN\_OPCODE enumeration (d3dcsx.h)

Article 02/22/2024

Scan opcodes.

## Syntax

C++

```
typedef enum D3DX11_SCAN_OPCODE {
    D3DX11_SCAN_OPCODE_ADD = 1,
    D3DX11_SCAN_OPCODE_MIN,
    D3DX11_SCAN_OPCODE_MAX,
    D3DX11_SCAN_OPCODE_MUL,
    D3DX11_SCAN_OPCODE_AND,
    D3DX11_SCAN_OPCODE_OR,
    D3DX11_SCAN_OPCODE_XOR
};
```

## Constants

[+] Expand table

`D3DX11_SCAN_OPCODE_ADD`

Value: 1

Add values.

`D3DX11_SCAN_OPCODE_MIN`

Take the minimum value.

`D3DX11_SCAN_OPCODE_MAX`

Take the maximum value.

`D3DX11_SCAN_OPCODE_MUL`

Multiply the values.

`D3DX11_SCAN_OPCODE_AND`

Perform a logical AND on the values.

`D3DX11_SCAN_OPCODE_OR`

Perform a logical OR on the values.

#### D3DX11\_SCAN\_OPCODE\_XOR

Perform a logical XOR on the values.

## Requirements

 Expand table

Requirement	Value
Header	d3dcsx.h

## See also

[D3DCSX 11 Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# D3DX 11 Reference

Article • 06/08/2021

The D3DX 11 API is described in this section.

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8, and is not supported for Microsoft Store apps. For details on recommended replacements for D3DX11, see [Living without D3DX](#). For older projects, you can make use of the [Microsoft.DXSDK.D3DX](#) NuGet package rather than relying on the legacy DirectX SDK or DirectSetup.

## In this section

↔ [Expand table](#)

Topic	Description
<a href="#">D3DX Interfaces</a>	This section contains reference information for the component object model (COM) interfaces provided by the D3DX utility library.
<a href="#">D3DX Functions</a>	This section contains information about the D3DX 11 functions.
<a href="#">D3DX Structures</a>	This section contains information about the D3DX 11 structures.
<a href="#">D3DX Enumerations</a>	This section contains information about D3DX 11 enumerations.

## Related topics

- [Direct3D 11 Reference](#)
- [Direct3D 11 Reference](#)
- 

## Feedback

Was this page helpful?

 Yes

 No



# D3DX Interfaces (Direct3D 11 Graphics)

Article • 08/18/2023

This section contains reference information for the component object model (COM) interfaces provided by the D3DX utility library.

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## In this section

Topic	Description
<a href="#">ID3DX11DataLoader</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Data loading object used by <a href="#">ID3DX11ThreadPump Interface</a> for loading data asynchronously.</p>
<a href="#">ID3DX11DataProcessor</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Data processing object used by <a href="#">ID3DX11ThreadPump Interface</a> for loading data asynchronously.</p>
<a href="#">ID3DX11ThreadPump</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>A thread pump executes tasks asynchronously. It is created by calling <a href="#">D3DX11CreateThreadPump</a>. There are several APIs that take an optional thread pump as a parameter, such as <a href="#">D3DX11CreateTextureFromFile</a> and <a href="#">D3DX11CompileFromFile</a>; if you pass a thread pump interface into these APIs, the functions will execute asynchronously on a separate thread. Particularly on multiprocessor machines, a thread pump can load resources and process data without a noticeable decrease in performance.</p>

## Related topics

[D3DX 11 Reference](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11DataLoader interface

Article • 08/18/2023

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Data loading object used by [ID3DX11ThreadPump Interface](#) for loading data asynchronously.

## Members

The **ID3DX11DataLoader** interface inherits from the [IUnknown](#) interface.

**ID3DX11DataLoader** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11DataLoader** interface has these methods.

Method	Description
<a href="#">Decompress</a>	<b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. Decompresses encoded data.
<a href="#">Destroy</a>	<b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. Destroys the loader after a work item completes.
<a href="#">Load</a>	<b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. Loads data from a disk.

## Remarks

This object can be inherited and its members redefined. Doing so would enable you to customize the API for loading and decompressing your own custom file formats.

# Requirements

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11DataLoader::Decompress method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Decompresses encoded data.

## Syntax

C++

```
HRESULT Decompress(
    [out] void    **ppData,
    [in]  SIZE_T *pcBytes
);
```

## Parameters

*ppData* [out]

Type: **void\*\***

Pointer to the raw data to decompress.

*pcBytes* [in]

Type: **SIZE\_T\***

The size of the data pointed to by *ppData*.

## Return value

Type: **HRESULT** ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

Use this method to load resources from file systems, such as ZIP files. When loading from an uncompressed resource, the decompression stage does not have to do any work.

[ID3DX11DataLoader Interface](#) can be inherited and its members redefined to support custom file formats.

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11DataLoader](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11DataLoader::Destroy method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Destroys the loader after a work item completes.

## Syntax

C++

```
HRESULT Destroy();
```

## Parameters

This method has no parameters.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

This method is used by an [ID3DX11ThreadPump Interface](#).

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11DataLoader](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11DataLoader::Load method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Loads data from a disk.

## Syntax

C++

```
HRESULT Load();
```

## Parameters

This method has no parameters.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

This method is used by an [ID3DX11ThreadPump Interface](#).

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11DataLoader](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11DataProcessor interface

Article • 08/18/2023

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Data processing object used by [ID3DX11ThreadPump Interface](#) for loading data asynchronously.

## Members

The **ID3DX11DataProcessor** interface inherits from the [IUnknown](#) interface.

**ID3DX11DataProcessor** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11DataProcessor** interface has these methods.

Method	Description
<a href="#">CreateDeviceObject</a>	<b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. Creates a device object.
<a href="#">Destroy</a>	<b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. Destroys the processor after a work item completes.
<a href="#">Process</a>	<b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. Processes data.

## Remarks

This object can be inherited and its members redefined for processing custom file formats.

# Requirements

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11DataProcessor::CreateDeviceObject method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Creates a device object.

## Syntax

C++

```
HRESULT CreateDeviceObject(  
    [out] void **ppDataObject  
);
```

## Parameters

*ppDataObject* [out]

Type: **void\*\***

Address of a pointer to the created device object.

## Return value

Type: **HRESULT** ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

This method is used by an [ID3DX11ThreadPump Interface](#).

# Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11DataProcessor](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11DataProcessor::Destroy method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Destroys the processor after a work item completes.

## Syntax

C++

```
HRESULT Destroy();
```

## Parameters

This method has no parameters.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

This method is used by an [ID3DX11ThreadPump Interface](#).

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11DataProcessor](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11DataProcessor::Process method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Processes data.

## Syntax

C++

```
HRESULT Process(
    [in] void     *pData,
    [in] SIZE_T   cBytes
);
```

## Parameters

*pData* [in]

Type: **void\***

Pointer to the data to be processed.

*cBytes* [in]

Type: **SIZE\_T**

Size of the data to be processed.

## Return value

Type: **HRESULT** ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

This method is used by an [ID3DX11ThreadPump Interface](#).

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11DataProcessor](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11ThreadPump interface

Article • 08/18/2023

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

A thread pump executes tasks asynchronously. It is created by calling [D3DX11CreateThreadPump](#). There are several APIs that take an optional thread pump as a parameter, such as [D3DX11CreateTextureFromFile](#) and [D3DX11CompileFromFile](#); if you pass a thread pump interface into these APIs, the functions will execute asynchronously on a separate thread. Particularly on multiprocessor machines, a thread pump can load resources and process data without a noticeable decrease in performance.

## Members

The ID3DX11ThreadPump interface inherits from the [IUnknown](#) interface.

ID3DX11ThreadPump also has these types of members:

- [Methods](#)

## Methods

The ID3DX11ThreadPump interface has these methods.

Method	Description
<a href="#">AddWorkItem</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Adds a work item to the thread pump.</p>
<a href="#">GetQueueStatus</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Gets the number of items in each of the three queues inside the thread pump.</p>
<a href="#">GetWorkItemCount</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store</p>

Method	Description
	<p>apps.</p> <p>Gets the number of work items in the thread pump.</p>
<a href="#">ProcessDeviceWorkItems</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Sets work items to the device after they have finished loading and processing.</p>
<a href="#">PurgeAllItems</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Clears all work items from the thread pump.</p>
<a href="#">WaitForAllItems</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Waits for all work items in the thread pump to finish.</p>

## Remarks

### Using a Thread Pump

The thread pump loads and processes data using the following three-step process:

1. Load and decompress the data with a [Data Loader](#). The data loader object has three methods that the thread pump will call internally as it is loading and decompressing the data: [ID3DX11DataLoader::Load](#), [ID3DX11DataLoader::Decompress](#), and [ID3DX11DataLoader::Destroy](#). The specific functionality of these three APIs differs depending on the type of data being loaded and decompressed. The data loader interface can also be inherited and its APIs can be changed if one is loading a data file defined in one's own custom format.
2. Process the data with a [Data Processor](#). The data processor object has three methods that the thread pump will call internally as it is processing the data: [ID3DX11DataProcessor::Process](#), [ID3DX11DataProcessor::CreateDeviceObject](#), and [ID3DX11DataProcessor::Destroy](#). The way it processes the data will be different depending on the type of data. For example, if the data is a texture stored as a JPEG, then [ID3DX11DataProcessor::Process](#) will do JPEG decompression to get the image's raw image bits. If the data is a shader, then [ID3DX11DataProcessor::Process](#) will compile the HLSL into bytecode. After the data has been processed a device object will be created for that data (with

[ID3DX11DataProcessor::CreateDeviceObject](#)) and the object will be added to a queue of device objects. The data processor interface can also be inherited and its APIs can be changed if one is processing a data file defined in one's own custom format.

3. Bind the device object to the device. This is done when one's application calls [ID3DX11ThreadPump::ProcessDeviceWorkItems](#), which will bind a specified number of objects in the queue of device objects to the device.

The thread pump can be used to load data in one of two ways: by calling an API that takes a thread pump as a parameter, such as [D3DX11CreateTextureFromFile](#) and [D3DX11CompileFromFile](#), or by calling [ID3DX11ThreadPump::AddWorkItem](#). In the case of the APIs that take a thread pump, the data loader and data processor are created internally. In the case of AddWorkItem, the data loader and data processor must be created beforehand and are then passed into AddWorkItem. D3DX11 provides a set of APIs for creating data loaders and data processors that have functionality for loading and processing common data formats. For custom data formats, the data loader and data processor interfaces must be inherited and their methods must be redefined.

The thread pump object takes up a substantial amount of resources, so generally only one should be created per application.

## Requirements

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[D3DX Interfaces](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3DX11ThreadPump::AddWorkItem method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Adds a work item to the thread pump.

## Syntax

C++

```
HRESULT AddWorkItem(
    [in] ID3DX11DataLoader     *pDataLoader,
    [in] ID3DX11DataProcessor *pDataProcessor,
    [in] HRESULT              *pHResult,
    [out] void                **ppDeviceObject
);
```

## Parameters

*pDataLoader* [in]

Type: [ID3DX11DataLoader\\*](#)

The loader that the thread pump will use when a work item requires data to be loaded.

*pDataProcessor* [in]

Type: [ID3DX11DataProcessor\\*](#)

The processor that the thread pump will use when a work item requires data to be processed.

*pHResult* [in]

Type: [HRESULT](#) ↴ \*

A pointer to the return value. May be **NULL**.

*ppDeviceObject* [out]

Type: **void\*\***

The device that uses the object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11ThreadPump](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11ThreadPump::GetQueueStatus method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Gets the number of items in each of the three queues inside the thread pump.

## Syntax

C++

```
HRESULT GetQueueStatus(
    [in] UINT *pIoQueue,
    [in] UINT *pProcessQueue,
    [in] UINT *pDeviceQueue
);
```

## Parameters

*pIoQueue* [in]

Type: **UINT\***

Number of items in the I/O queue.

*pProcessQueue* [in]

Type: **UINT\***

Number of items in the process queue.

*pDeviceQueue* [in]

Type: **UINT\***

Number of items in the device queue.

# Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11ThreadPump](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11ThreadPump::GetWorkItemCount method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Gets the number of work items in the thread pump.

## Syntax

C++

```
UINT GetWorkItemCount();
```

## Parameters

This method has no parameters.

## Return value

Type: [UINT](#)

The number of work items queued in the thread pump.

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3DX11ThreadPump::ProcessDeviceWorkItems method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Sets work items to the device after they have finished loading and processing.

## Syntax

C++

```
HRESULT ProcessDeviceWorkItems(
    [in] UINT iWorkItemCount
);
```

## Parameters

*iWorkItemCount* [in]

Type: [UINT](#)

The number of work items to set to the device.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

When the thread pump has finished loading and processing a resource or shader, it will hold it in a queue until this API is called, at which point the processed items will be set

to the device. This is useful for controlling the amount of processing that is spent on binding resources to the device for each frame.

As an example of how one might use this API, say you are nearing the end of one level in your game and you want to begin preloading the textures, shaders, and other resources for the next level. The thread pump will begin loading, decompressing, and processing the resources and shaders on a separate thread until they are ready to be set to the device, at which point it will leave them in a queue. One may not want to set all the resources and shaders to the device at once because this may cause a noticeable temporary slow down in the game's performance. So, this API could be called once per frame so that only a small number work items would be set to the device on each frame, thereby spreading the work load of binding resources to the device over several frames.

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[ID3DX11ThreadPump](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11ThreadPump::PurgeAllItems method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Clears all work items from the thread pump.

## Syntax

C++

```
HRESULT PurgeAllItems();
```

## Parameters

This method has no parameters.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3DX11ThreadPump::WaitForAllItems method

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Waits for all work items in the thread pump to finish.

## Syntax

C++

```
HRESULT WaitForAllItems();
```

## Parameters

This method has no parameters.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3DX Functions (Direct3D 11 Graphics)

Article • 08/18/2023

This section contains information about the D3DX 11 functions.

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## In this section

Topic	Description
<a href="#">D3DX11CompileFromFile</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you compile offline by using the Fxc.exe command-line compiler or use one of the HLSL compile APIs, like the <a href="#">D3DCompileFromFile</a> API.</p> <p>Compile a shader or an effect from a file.</p>
<a href="#">D3DX11CompileFromMemory</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you compile offline by using the Fxc.exe command-line compiler or use one of the HLSL compile APIs, like the <a href="#">D3DCompile</a> API.</p> <p>Compile a shader or an effect that is loaded in memory.</p>
<a href="#">D3DX11CompileFromResource</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use <a href="#">resource functions</a>, then compile offline by using the Fxc.exe command-line compiler or use one</p>

Topic	Description
	<p>of the HLSL compile APIs, like the <a href="#">D3DCompile</a> API.</p> <p>Compile a shader or an effect from a resource.</p>
<a href="#">D3DX11ComputeNormalMap</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">DirectXTex</a> library, <a href="#">ComputeNormalMap</a>.</p> <p>Converts a height map into a normal map. The (x,y,z) components of each normal are mapped to the (r,g,b) channels of the output texture.</p>
<a href="#">D3DX11CreateAsyncCompilerProcessor</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create an asynchronous-data processor for a shader.</p>
<a href="#">D3DX11CreateAsyncFileLoader</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create an asynchronous-file loader.</p>
<a href="#">D3DX11CreateAsyncMemoryLoader</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create an asynchronous-memory loader.</p>
<a href="#">D3DX11CreateAsyncResourceLoader</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create an asynchronous-resource loader.</p>
<a href="#">D3DX11CreateAsyncShaderPreprocessProcessor</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create a data processor for a shader asynchronously.</p>

Topic	Description
<a href="#">D3DX11CreateAsyncTextureInfoProcessor</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create a data processor to be used with a <a href="#">thread pump</a>.</p>
<a href="#">D3DX11CreateAsyncTextureProcessor</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create a data processor to be used with a <a href="#">thread pump</a>.</p>
<a href="#">D3DX11CreateAsyncShaderResourceViewProcessor</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create a data processor that will load a resource and then create a shader-resource view for it. Data processors are a component of the asynchronous data loading feature in D3DX11 that uses <a href="#">thread pumps</a>.</p>
<a href="#">D3DX11CreateShaderResourceViewFromFile</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use these:</p> <ul style="list-style-type: none"> <li>- <a href="#">DirectXTK</a> library (runtime),</li> <li><a href="#">CreateXXXTextureFromFile</a> (where XXX is DDS or WIC)</li> <li>- <a href="#">DirectXToolbox</a> library (tools),</li> <li><a href="#">LoadFromXXXFile</a> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then</li> <li><a href="#">CreateShaderResourceView</a></li> </ul> <p>Create a shader-resource view from a file.</p>
<a href="#">D3DX11CreateShaderResourceViewFromMemory</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use these:</p>

Topic	Description
	<ul style="list-style-type: none"> <li>- <a href="#">DirectXTK</a> library (runtime),</li> <li><b>CreateXXXTextureFromMemory</b> (where XXX is DDS or WIC)</li> <li>- <a href="#">DirectXToolbox</a> library (tools),</li> <li><b>LoadFromXXXMemory</b> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then</li> <li><b>CreateShaderResourceView</b></li> <li>Create a shader-resource view from a file in memory.</li> </ul>
<a href="#">D3DX11CreateShaderResourceViewFromResource</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use <a href="#">resource functions</a>, then these:</p> <ul style="list-style-type: none"> <li>- <a href="#">DirectXTK</a> library (runtime),</li> <li><b>CreateXXXTextureFromMemory</b> (where XXX is DDS or WIC)</li> <li>- <a href="#">DirectXToolbox</a> library (tools),</li> <li><b>LoadFromXXXMemory</b> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then</li> <li><b>CreateShaderResourceView</b></li> <li>Create a shader-resource view from a resource.</li> </ul>
<a href="#">D3DX11CreateTextureFromFile</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use these:</p> <ul style="list-style-type: none"> <li>- <a href="#">DirectXTK</a> library (runtime),</li> <li><b>CreateXXXTextureFromFile</b> (where XXX is DDS or WIC)</li> <li>- <a href="#">DirectXToolbox</a> library (tools),</li> <li><b>LoadFromXXXFile</b> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then</li> <li><b>CreateTexture</b></li> <li>Create a texture resource from a file.</li> </ul>

Topic	Description
<a href="#">D3DX11CreateTextureFromMemory</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use these:</p> <ul style="list-style-type: none"> <li>- <a href="#">DirectXTK</a> library (runtime),</li> <li><a href="#">CreateXXXTextureFromMemory</a> (where XXX is DDS or WIC)</li> <li>- <a href="#">DirectXToolbox</a> library (tools),</li> <li><a href="#">LoadFromXXXMemory</a> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then <a href="#">CreateTexture</a></li> </ul> <p>Create a texture resource from a file residing in system memory.</p>
<a href="#">D3DX11CreateTextureFromResource</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use <a href="#">resource functions</a>, then these:</p> <ul style="list-style-type: none"> <li>- <a href="#">DirectXTK</a> library (runtime),</li> <li><a href="#">CreateXXXTextureFromMemory</a> (where XXX is DDS or WIC)</li> <li>- <a href="#">DirectXToolbox</a> library (tools),</li> <li><a href="#">LoadFromXXXMemory</a> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then <a href="#">CreateTexture</a></li> </ul> <p>Create a texture from another resource.</p>
<a href="#">D3DX11CreateThreadPump</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.</p> <p>Create a thread pump.</p>
<a href="#">D3DX11FilterTexture</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">DirectXToolbox</a></p>

Topic	Description
	<p>library, <a href="#">GenerateMipMaps</a> and <a href="#">GenerateMipMaps3D</a>. Generates mipmap chain using a particular texture filter.</p>
<a href="#">D3DX11GetImageInfoFromFile</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">DirectXTex</a> library, <a href="#">GetMetadataFromXXXFile</a> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games).</p> <p>Retrieves information about a given image file.</p>
<a href="#">D3DX11GetImageInfoFromMemory</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">DirectXTex</a> library, <a href="#">GetMetadataFromXXXMemory</a> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games).</p> <p>Get information about an image already loaded into memory.</p>
<a href="#">D3DX11GetImageInfoFromResource</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use <a href="#">resource functions</a>, then use <a href="#">DirectXTex</a> library (tools), <a href="#">LoadFromXXXMemory</a> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games).</p> <p>Retrieves information about a given image in a resource.</p>

Topic	Description
<a href="#">D3DX11LoadTextureFromTexture</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">DirectXTex</a> library, <b>Resize</b>, <b>Convert</b>, <b>Compress</b>, <b>Decompress</b>, and/or <b>CopyRectangle</b>.</p> <p>Load a texture from a texture.</p>
<a href="#">D3DX11PreprocessShaderFromFile</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">D3DPreprocess</a> API.</p> <p>Create a shader from a file without compiling it.</p>
<a href="#">D3DX11PreprocessShaderFromMemory</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">D3DPreprocess</a> API.</p> <p>Create a shader from memory without compiling it.</p>
<a href="#">D3DX11PreprocessShaderFromResource</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">D3DPreprocess</a> API.</p> <p>Create a shader from a resource without compiling it.</p>
<a href="#">D3DX11SaveTextureToFile</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">DirectXTex</a> library, <b>CaptureTexture</b> then <b>SaveToXXXFile</b> (where XXX is WIC, DDS, or TGA; WIC</p>

Topic	Description
	<p>doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games). For the simplified scenario of creating a screen shot from a render target texture, we recommend that you use the <a href="#">DirectXTK</a> library, <a href="#">SaveDDSTextureToFile</a> or <a href="#">SaveWICTextureToFile</a>. Save a texture to a file.</p>
<a href="#">D3DX11SaveTextureToMemory</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">DirectXTex</a> library, <a href="#">CaptureTexture</a> then <a href="#">SaveToXXXMemory</a> (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games).</p> <p>Save a texture to memory.</p>
<a href="#">D3DX11SHProjectCubeMap</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">Spherical Harmonics Math</a> library, <a href="#">SHProjectCubeMap</a>.</p> <p>Projects a function represented in a cube map into spherical harmonics.</p>
<a href="#">D3DX11UnsetAllDeviceObjects</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p><b>Note:</b> Instead of using this function, we recommend that you use the <a href="#">ID3D11DeviceContext::ClearState</a> method.</p> <p>Removes all resources from the device by setting their pointers to <b>NULL</b>. This should be called during shutdown of your application. It helps ensure that when one is releasing all of their resources that none of them are bound to the device.</p>

# Related topics

[D3DX 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CompileFromFile function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you compile offline by using the Fxc.exe command-line compiler or use one of the HLSL compile APIs, like the [D3DCompileFromFile](#) API.

Compile a shader or an effect from a file.

## Syntax

C++

```
HRESULT D3DX11CompileFromFile(  
    _In_          LPCTSTR           pSrcFile,  
    _In_  const D3D10_SHADER_MACRO *pDefines,  
    _In_          LPD3D10INCLUDE   pInclude,  
    _In_          LPCSTR            pFunctionName,  
    _In_          LPCSTR            pProfile,  
    _In_          UINT              Flags1,  
    _In_          UINT              Flags2,  
    _In_          ID3DX11ThreadPump *pPump,  
    _Out_         ID3D10Blob        **ppShader,  
    _Out_         ID3D10Blob        **ppErrorMsgs,  
    _Out_         HRESULT           *pHResult  
) ;
```

## Parameters

*pSrcFile* [in]

Type: [LPCTSTR](#)

The name of the file that contains the shader code. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*pDefines* [in]

Type: [const D3D10\\_SHADER\\_MACRO\\*](#)

Optional. Pointer to an array of macro definitions (see [D3D10\\_SHADER\\_MACRO](#)). The last structure in the array serves as a terminator and must have all members set to 0. If not used, set *pDefines* to **NULL**.

*pInclude* [in]

Type: [LPD3D10INCLUDE](#)

Optional. Pointer to an interface for handling include files. Setting this to **NULL** will cause a compile error if a shader contains a #include.

*pFunctionName* [in]

Type: [LPCSTR](#)

Name of the shader-entry point function where shader execution begins. When you compile an effect, [D3DX11CompileFromFile](#) ignores *pFunctionName*; we recommend that you set *pFunctionName* to **NULL** because it is good programming practice to set a pointer parameter to **NULL** if the called function will not use it.

*pProfile* [in]

Type: [LPCSTR](#)

A string that specifies the shader model; can be any profile in shader model 2, shader model 3, shader model 4, or shader model 5. The profile can also be for effect type (for example, fx\_4\_1).

*Flags1* [in]

Type: [UINT](#)

Shader [compile flags](#).

*Flags2* [in]

Type: [UINT](#)

Effect **compile flags**. When you compile a shader and not an effect file, **D3DX11CompileFromFile** ignores *Flags2*; we recommend that you set *Flags2* to zero because it is good programming practice to set a nonpointer parameter to zero if the called function will not use it.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). Use **NULL** to specify that this function should not return until it is completed.

*ppShader* [out]

Type: [ID3D10Blob\\*\\*](#)

A pointer to memory which contains the compiled shader, as well as any embedded debug and symbol-table information.

*ppErrorMsgs* [out]

Type: [ID3D10Blob\\*\\*](#)

A pointer to memory which contains a listing of errors and warnings that occurred during compilation. These errors and warnings are identical to the debug output from a debugger.

*pHResult* [out]

Type: [HRESULT](#) ↗\*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#) ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

**D3DX11CompileFromFile** returns **E\_INVALIDARG** if you supply a non-**NULL** value to the *pHResult* parameter when you supply **NULL** to the *pPump* parameter. For more information about this situation, see Remarks.

## Remarks

For more information about **D3DX11CompileFromFile**, see **D3DCompile**.

You must supply **NULL** to the *pHResult* parameter if you also supply **NULL** to the *pPump* parameter. Otherwise, you cannot create a shader by using the compiled shader code that **D3DX11CompileFromFile** returns in the memory that the *ppShader* parameter points to. To create a shader from complied shader code, you call one of the following **ID3D11Device** interface methods:

- [CreateComputeShader](#)
- [CreateDomainShader](#)
- [CreateGeometryShader](#)
- [CreateGeometryShaderWithStreamOutput](#)
- [CreateHullShader](#)
- [CreatePixelShader](#)
- [CreateVertexShader](#)

In addition, if you supply a non-**NULL** value to *pHResult* when you supply **NULL** to *pPump*, **D3DX11CompileFromFile** returns the **E\_INVALIDARG** error code.

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CompileFromMemory function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you compile offline by using the Fxc.exe command-line compiler or use one of the HLSL compile APIs, like the [D3DCompile API](#).

Compile a shader or an effect that is loaded in memory.

## Syntax

C++

```
HRESULT D3DX11CompileFromMemory(
    _In_      LPCSTR          pSrcData,
    _In_      SIZE_T           SrcDataLen,
    _In_      LPCSTR          pFileName,
    _In_      const D3D10_SHADER_MACRO *pDefines,
    _In_      LPD3D10INCLUDE    pInclude,
    _In_      LPCSTR          pFunctionName,
    _In_      LPCSTR          pProfile,
    _In_      UINT             Flags1,
    _In_      UINT             Flags2,
    _In_      ID3DX11ThreadPump *pPump,
    _Out_     ID3D10Blob       **ppShader,
    _Out_     ID3D10Blob       **ppErrorMsgs,
    _Out_     HRESULT          *pHResult
);
```

## Parameters

*pSrcData* [in]

Type: [LPCSTR](#)

Pointer to the shader in memory.

*SrcDataLen* [in]

Type: [SIZE\\_T](#)

Size of the shader in memory.

*pFileName* [in]

Type: [LPCSTR](#)

The name of the file that contains the shader code.

*pDefines* [in]

Type: [const D3D10\\_SHADER\\_MACRO\\*](#)

Optional. Pointer to an array of macro definitions (see [D3D10\\_SHADER\\_MACRO](#)). The last structure in the array serves as a terminator and must have all members set to 0. If not used, set *pDefines* to **NULL**.

*pInclude* [in]

Type: [LPD3D10INCLUDE](#)

Optional. Pointer to an interface for handling include files. Setting this to **NULL** will cause a compile error if a shader contains a #include.

*pFunctionName* [in]

Type: [LPCSTR](#)

Name of the shader-entry point function where shader execution begins. When you compile an effect, [D3DX11CompileFromMemory](#) ignores *pFunctionName*; we recommend that you set *pFunctionName* to **NULL** because it is good programming practice to set a pointer parameter to **NULL** if the called function will not use it.

*pProfile* [in]

Type: [LPCSTR](#)

A string that specifies the shader model; can be any profile in shader model 2, shader model 3, shader model 4, or shader model 5. The profile can also be for effect type (for example, fx\_4\_1).

*Flags1* [in]

Type: [UINT](#)

Shader [compile flags](#).

*Flags2* [in]

Type: [UINT](#)

Effect [compile flags](#). When you compile a shader and not an effect file, [D3DX11CompileFromMemory](#) ignores *Flags2*; we recommend that you set *Flags2* to zero because it is good programming practice to set a nonpointer parameter to zero if the called function will not use it.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). Use **NULL** to specify that this function should not return until it is completed.

*ppShader* [out]

Type: [ID3D10Blob\\*\\*](#)

A pointer to memory which contains the compiled shader, as well as any embedded debug and symbol-table information.

*ppErrorMsgs* [out]

Type: [ID3D10Blob\\*\\*](#)

A pointer to memory which contains a listing of errors and warnings that occurred during compilation. These errors and warnings are identical to the debug output from a debugger.

*pHResult* [out]

Type: [HRESULT](#) \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#) ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

`D3DX11CompileFromMemory` returns `E_INVALIDARG` if you supply non-**NULL** to the *pHResult* parameter when you supply **NULL** to the *pPump* parameter. For more information about this situation, see Remarks.

## Remarks

For more information about `D3DX11CompileFromMemory`, see [D3DCompile](#).

You must supply **NULL** to the *pHResult* parameter if you also supply **NULL** to the *pPump* parameter. Otherwise, you cannot subsequently create a shader by using the compiled shader code that `D3DX11CompileFromMemory` returns in the memory that the *ppShader* parameter points to. To create a shader from compiled shader code, you call one of the following [ID3D11Device](#) interface methods:

- [CreateComputeShader](#)
- [CreateDomainShader](#)
- [CreateGeometryShader](#)
- [CreateGeometryShaderWithStreamOutput](#)
- [CreateHullShader](#)
- [CreatePixelShader](#)
- [CreateVertexShader](#)

In addition, if you supply a non-**NULL** value to *pHResult* when you supply **NULL** to *pPump*, `D3DX11CompileFromMemory` returns the `E_INVALIDARG` error code.

## Requirements

Requirement	Value
Header	<code>D3DX11async.h</code>
Library	<code>D3DX11.lib</code>

## See also

[D3DX Functions](#)

## Feedback



Was this page helpful?  Yes  No

Get help at Microsoft Q&A

# D3DX11CompileFromResource function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use [resource functions](#), then compile offline by using the Fxc.exe command-line compiler or use one of the HLSL compile APIs, like the [D3DCompile API](#).

Compile a shader or an effect from a resource.

## Syntax

C++

```
HRESULT D3DX11CompileFromResource(
    _In_          HMODULE           hSrcModule,
    _In_          LPCTSTR            pSrcResource,
    _In_          LPCTSTR            pSrcFileName,
    _In_  const D3D10_SHADER_MACRO *pDefines,
    _In_          LPD3D10INCLUDE     pInclude,
    _In_          LPCSTR             pFunctionName,
    _In_          LPCSTR             pProfile,
    _In_          UINT               Flags1,
    _In_          UINT               Flags2,
    _In_          ID3DX11ThreadPump *pPump,
    _Out_         ID3D10Blob        **ppShader,
    _Out_         ID3D10Blob        **ppErrorMsgs,
    _Out_         HRESULT            *pHResult
);
```

## Parameters

*hSrcModule* [in]

Type: [HMODULE](#)

Handle to the resource module containing the shader. HMODULE can be obtained with [GetModuleHandle Function](#).

*pSrcResource* [in]

Type: [LPCTSTR](#)

Name of the resource containing the shader. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*pSrcFileName* [in]

Type: [LPCTSTR](#)

Optional. Effect file name, which is used for error messages only. Can be **NULL**.

*pDefines* [in]

Type: [const D3D10\\_SHADER\\_MACRO\\*](#)

Optional. Pointer to an array of macro definitions (see [D3D10\\_SHADER\\_MACRO](#)). The last structure in the array serves as a terminator and must have all members set to 0. If not used, set *pDefines* to **NULL**.

*pInclude* [in]

Type: [LPD3D10INCLUDE](#)

Optional. Pointer to an interface for handling include files. Setting this to **NULL** will cause a compile error if a shader contains a #include.

*pFunctionName* [in]

Type: [LPCSTR](#)

Name of the shader-entry point function where shader execution begins. When you compile an effect, [D3DX11CompileFromResource](#) ignores *pFunctionName*; we recommend that you set *pFunctionName* to **NULL** because it is good programming practice to set a pointer parameter to **NULL** if the called function will not use it.

*pProfile* [in]

Type: [LPCSTR](#)

A string that specifies the shader model; can be any profile in shader model 2, shader model 3, shader model 4, or shader model 5. The profile can also be for effect type (for

example, `fx_4_1`).

*Flags1* [in]

Type: **UINT**

Shader [compile flags](#).

*Flags2* [in]

Type: **UINT**

Effect [compile flags](#). When you compile a shader and not an effect file, `D3DX11CompileFromResource` ignores *Flags2*; we recommend that you set *Flags2* to zero because it is good programming practice to set a nonpointer parameter to zero if the called function will not use it.

*pPump* [in]

Type: [\*\*ID3DX11ThreadPump\*\*](#)\*

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). Use **NULL** to specify that this function should not return until it is completed.

*ppShader* [out]

Type: [\*\*ID3D10Blob\*\*](#)\*\*

A pointer to memory which contains the compiled shader, as well as any embedded debug and symbol-table information.

*ppErrorMsgs* [out]

Type: [\*\*ID3D10Blob\*\*](#)\*\*

A pointer to memory which contains a listing of errors and warnings that occurred during compilation. These errors and warnings are identical to the debug output from a debugger.

*pHResult* [out]

Type: [\*\*HRESULT\*\*](#) ↴ \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

**D3DX11CompileFromResource** returns **E\_INVALIDARG** if you supply non-**NULL** to the *pHResult* parameter when you supply **NULL** to the *pPump* parameter. For more information about this situation, see Remarks.

## Remarks

For more information about **D3DX11CompileFromResource**, see [D3DCompile](#).

You must supply **NULL** to the *pHResult* parameter if you also supply **NULL** to the *pPump* parameter. Otherwise, you cannot subsequently create a shader by using the compiled shader code that **D3DX11CompileFromResource** returns in the memory that the *ppShader* parameter points to. To create a shader from compiled shader code, you call one of the following [ID3D11Device](#) interface methods:

- [CreateComputeShader](#)
- [CreateDomainShader](#)
- [CreateGeometryShader](#)
- [CreateGeometryShaderWithStreamOutput](#)
- [CreateHullShader](#)
- [CreatePixelShader](#)
- [CreateVertexShader](#)

In addition, if you supply a non-**NULL** value to *pHResult* when you supply **NULL** to *pPump*, **D3DX11CompileFromResource** returns the **E\_INVALIDARG** error code.

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11ComputeNormalMap function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [DirectXTex](#) library, `ComputeNormalMap`.

Converts a height map into a normal map. The (x,y,z) components of each normal are mapped to the (r,g,b) channels of the output texture.

## Syntax

C++

```
HRESULT D3DX11ComputeNormalMap(
    _In_ ID3D11DeviceContext *pContext,
    _In_ ID3D11Texture2D     *pSrcTexture,
    _In_ UINT                 Flags,
    _In_ UINT                 Channel,
    _In_ FLOAT                Amplitude,
    _In_ ID3D11Texture2D     *pDestTexture
);
```

## Parameters

*pContext* [in]

Type: [ID3D11DeviceContext\\*](#)

Pointer to an [ID3D11DeviceContext](#) interface, representing the source height-map texture.

*pSrcTexture* [in]

Type: [ID3D11Texture2D\\*](#)

Pointer to an [ID3D11Texture2D](#) interface, representing the source height-map texture.

*Flags* [in]

Type: [UINT](#)

One or more D3DX\_NORMALMAP flags that control generation of normal maps.

*Channel* [in]

Type: [UINT](#)

One D3DX\_CHANNEL flag specifying the source of height information.

*Amplitude* [in]

Type: [FLOAT](#)

Constant value multiplier that increases (or decreases) the values in the normal map.  
Higher values usually make bumps more visible, lower values usually make bumps less visible.

*pDestTexture* [in]

Type: [ID3D11Texture2D\\*](#)

Pointer to an [ID3D11Texture2D](#) interface, representing the destination texture.

## Return value

Type: [HRESULT](#) ↗

If the function succeeds, the return value is D3D\_OK. If the function fails, the return value can be the following value: D3DERR\_INVALIDCALL.

## Remarks

This method computes the normal by using the central difference with a kernel size of 3x3. RGB channels in the destination contain biased (x,y,z) components of the normal.  
The central differencing denominator is hardcoded to 2.0.

## Requirements

Requirement	Value
-------------	-------

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateAsyncCompilerProcessor function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create an asynchronous-data processor for a shader.

## Syntax

C++

```
HRESULT D3DX11CreateAsyncCompilerProcessor(
    _In_          LPCSTR           pFileName,
    _In_  const D3D11_SHADER_MACRO *pDefines,
    _In_          LPD3D10INCLUDE   pInclude,
    _In_          LPCSTR           pFunctionName,
    _In_          LPCSTR           pProfile,
    _In_          UINT             Flags1,
    _In_          UINT             Flags2,
    _Out_         ID3D10Blob      **ppCompiledShader,
    _Out_         ID3D10Blob      **ppErrorBuffer,
    _Out_         ID3DX11DataProcessor **ppDataProcessor
);
```

## Parameters

*pFileName* [in]

Type: [LPCSTR](#)

A string that contains the shader filename.

*pDefines* [in]

Type: [const D3D11\\_SHADER\\_MACRO\\*](#)

A NULL-terminated array of shader macros; set this to **NULL** to specify no macros.

*pInclude* [in]

Type: [LPD3D10INCLUDE](#)

A pointer to an include interface. This parameter can be **NULL**.

*pFunctionName* [in]

Type: [LPCSTR](#)

Name of the shader-entry point function where shader execution begins. When you compile an effect, [D3DX11CreateAsyncCompilerProcessor](#) ignores *pFunctionName*; we recommend that you set *pFunctionName* to **NULL** because it is good programming practice to set a pointer parameter to **NULL** if the called function will not use it..

*pProfile* [in]

Type: [LPCSTR](#)

A string that specifies the shader profile or shader model; can be any profile in shader model 2, shader model 3, shader model 4, or shader model 5. The profile can also be for effect type (for example, fx\_4\_1).

*Flags1* [in]

Type: [UINT](#)

Shader compile flags.

*Flags2* [in]

Type: [UINT](#)

Effect compile flags. When you compile a shader and not an effect file, [D3DX11CreateAsyncCompilerProcessor](#) ignores *Flags2*; we recommend that you set *Flags2* to zero because it is good programming practice to set a nonpointer parameter to zero if the called function will not use it.

*ppCompiledShader* [out]

Type: [ID3D10Blob\\*\\*](#)

Address of a pointer to the compiled effect.

*ppErrorBuffer* [out]

Type: [ID3D10Blob\\*\\*](#)

Address of a pointer to compile errors.

*ppDataProcessor* [out]

Type: [ID3DX11DataProcessor\\*\\*](#)

Address of a pointer to a buffer that contains the data processor created (see [ID3DX11DataProcessor Interface](#)).

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

There s no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#)) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

## Feedback



Was this page helpful? [!\[\]\(84678247d8db42387a6086a0c381ccde\_img.jpg\) Yes](#) | [!\[\]\(10cacc74709c864c41a12d67867e05cd\_img.jpg\) No](#)

Get help at Microsoft Q&A

# D3DX11CreateAsyncFileLoader function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create an asynchronous-file loader.

## Syntax

C++

```
HRESULT D3DX11CreateAsyncFileLoader(
    _In_     LPCTSTR             pFileName,
    _Out_    ID3DX11DataLoader **ppDataLoader
);
```

## Parameters

*pFileName* [in]

Type: [LPCTSTR](#)

The name of the file to load. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*ppDataLoader* [out]

Type: [ID3DX11DataLoader\\*\\*](#)

Address of a pointer to the asynchronous-data loader (see [ID3DX11DataLoader Interface](#)).

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

There s no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#)) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateAsyncMemoryLoader function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create an asynchronous-memory loader.

## Syntax

C++

```
HRESULT D3DX11CreateAsyncMemoryLoader(
    _In_    LPCVOID          pData,
    _In_    SIZE_T           cbData,
    _Out_   ID3DX11DataLoader **ppDataLoader
);
```

## Parameters

*pData* [in]

Type: [LPCVOID](#)

Pointer to the data.

*cbData* [in]

Type: [SIZE\\_T](#)

Size of the data.

*ppDataLoader* [out]

Type: [ID3DX11DataLoader\\*\\*](#)

The address of a pointer to the asynchronous-data loader (see [ID3DX11DataLoader Interface](#)).

# Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

There's no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#)) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateAsyncResourceLoader function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create an asynchronous-resource loader.

## Syntax

C++

```
HRESULT D3DX11CreateAsyncResourceLoader(
    _In_ HMODULE hSrcModule,
    _In_ LPCTSTR pSrcResource,
    _Out_ ID3DX11DataLoader **ppDataLoader
);
```

## Parameters

*hSrcModule* [in]

Type: [HMODULE](#)

Handle to the resource module. Use [GetModuleHandle Function](#) to get the handle.

*pSrcResource* [in]

Type: [LPCTSTR](#)

Name of the resource in *hSrcModule*. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*ppDataLoader* [out]

Type: [ID3DX11DataLoader\\*\\*](#)

The address of a pointer to the asynchronous-data loader (see [ID3DX11DataLoader Interface](#)).

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

There's no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#)) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateAsyncShaderPreprocessProcessor function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create a data processor for a shader asynchronously.

## Syntax

C++

```
HRESULT D3DX11CreateAsyncShaderPreprocessProcessor(
    _In_          LPCSTR           pFileName,
    _In_  const D3D11_SHADER_MACRO *pDefines,
    _In_          LPD3D10INCLUDE   pInclude,
    _Out_         ID3D10Blob       **ppShaderText,
    _Out_         ID3D10Blob       **ppErrorBuffer,
    _Out_         ID3DX11DataProcessor **ppDataProcessor
);
```

## Parameters

*pFileName* [in]

Type: [LPCSTR](#)

A string that contains the shader filename.

*pDefines* [in]

Type: [const D3D11\\_SHADER\\_MACRO\\*](#)

A NULL-terminated array of shader macros; set this to **NULL** to specify no macros.

*pInclude* [in]

Type: [LPD3D10INCLUDE](#)

A pointer to an include interface; set this to **NULL** to specify there is no include file.

*ppShaderText* [out]

Type: **ID3D10Blob\*\***

Address of a pointer to a buffer that contains the ASCII text of the shader.

*ppErrorBuffer* [out]

Type: **ID3D10Blob\*\***

Address of a pointer to a buffer that contains compile errors.

*ppDataProcessor* [out]

Type: **ID3DX11DataProcessor\*\***

Address of a pointer to a buffer that contains the data processor created (see [ID3DX11DataProcessor Interface](#)).

## Return value

Type: **HRESULT** ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

There s no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#) ↗) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

---

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateAsyncTextureInfoProcessor function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create a data processor to be used with a [thread pump](#).

## Syntax

C++

```
HRESULT D3DX11CreateAsyncTextureInfoProcessor(
    _In_    D3DX11_IMAGE_INFO     *pImageInfo,
    _Out_   ID3DX11DataProcessor **ppDataProcessor
);
```

## Parameters

*pImageInfo* [in]

Type: [D3DX11\\_IMAGE\\_INFO\\*](#)

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*ppDataProcessor* [out]

Type: [ID3DX11DataProcessor\\*\\*](#)

Address of a pointer to a buffer that contains the data processor created (see [ID3DX11DataProcessor Interface](#)).

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

This API does creates a data-processor interface; [D3DX11CreateAsyncTextureProcessor](#) creates the data-processor interface and loads the texture.

There s no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#)) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateAsyncTextureProcessor function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create a data processor to be used with a [thread pump](#).

## Syntax

C++

```
HRESULT D3DX11CreateAsyncTextureProcessor(
    _In_ ID3D11Device           *pDevice,
    _In_ D3DX11_IMAGE_LOAD_INFO *pCreateInfo,
    _Out_ ID3DX11DataProcessor   **ppDataProcessor
);
```

## Parameters

*pDevice* [in]

Type: [ID3D11Device](#)\*

A pointer to the device (see [ID3D11Device](#)).

*pCreateInfo* [in]

Type: [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)\*

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*ppDataProcessor* [out]

Type: [ID3DX11DataProcessor](#)\*\*

Address of a pointer to a buffer that contains the data processor created (see [ID3DX11DataProcessor Interface](#)).

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

This API does creates a data-processor interface and loads the texture; [D3DX11CreateAsyncTextureInfoProcessor](#) creates the data-processor interface.

There s no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#)) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateAsyncShaderResourceViewProcessor function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create a data processor that will load a resource and then create a shader-resource view for it. Data processors are a component of the asynchronous data loading feature in D3DX11 that uses [thread pumps](#).

## Syntax

C++

```
HRESULT D3DX11CreateAsyncShaderResourceViewProcessor(
    _In_    ID3D11Device           *pDevice,
    _In_    D3DX11_IMAGE_LOAD_INFO *pLoadInfo,
    _Out_   ID3DX11DataProcessor  **ppDataProcessor
);
```

## Parameters

*pDevice* [in]

Type: [ID3D11Device\\*](#)

Pointer to the Direct3D device (see [ID3D11Device](#)) that will be used to create a resource and a shader-resource view for that resource.

*pLoadInfo* [in]

Type: [D3DX11\\_IMAGE\\_LOAD\\_INFO\\*](#)

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*ppDataProcessor* [out]

Type: [ID3DX11DataProcessor\\*\\*](#)

Address of a pointer to a buffer that contains the data processor created (see [ID3DX11DataProcessor Interface](#)).

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

There s no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#)) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3DX11CreateShaderResourceViewFromFile function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use these:

- [DirectXTK](#) library (runtime), [CreateXXXTextureFromFile](#) (where XXX is DDS or WIC)
- [DirectXTex](#) library (tools), [LoadFromXXXFile](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then [CreateShaderResourceView](#)

Create a shader-resource view from a file.

## Syntax

C++

```
HRESULT D3DX11CreateShaderResourceViewFromFile(
    _In_ ID3D11Device                  *pDevice,
    _In_ LPCTSTR                      pSrcFile,
    _In_ D3DX11_IMAGE_LOAD_INFO      *pCreateInfo,
    _In_ ID3DX11ThreadPump           *pPump,
    _Out_ ID3D11ShaderResourceView **ppShaderResourceView,
    _Out_ HRESULT                     *pHResult
);
```

## Parameters

*pDevice* [in]

Type: [ID3D11Device\\*](#)

A pointer to the device (see [ID3D11Device](#)) that will use the resource.

*pSrcFile* [in]

Type: [LPCTSTR](#)

Name of the file that contains the shader-resource view. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*pLoadInfo* [in]

Type: [D3DX11\\_IMAGE\\_LOAD\\_INFO\\*](#)

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

Pointer to a thread-pump interface (see [ID3DX11ThreadPump Interface](#)). If **NULL** is specified, this function will behave synchronously and will not return until it is finished.

*ppShaderResourceView* [out]

Type: [ID3D11ShaderResourceView\\*\\*](#)

Address of a pointer to the shader-resource view (see [ID3D11ShaderResourceView](#)).

*pHResult* [out]

Type: [HRESULT](#) \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#) ↴

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

For a list of supported image formats, see [D3DX11\\_IMAGE\\_FILE\\_FORMAT](#).

# Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateShaderResourceViewFromMemory function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use these:

- [DirectXTK](#) library (runtime), [CreateXXXTextureFromMemory](#) (where XXX is DDS or WIC)
- [DirectXTex](#) library (tools), [LoadFromXXXMemory](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then [CreateShaderResourceView](#)

Create a shader-resource view from a file in memory.

## Syntax

C++

```
HRESULT D3DX11CreateShaderResourceViewFromMemory(
    _In_     ID3D11Device                  *pDevice,
    _In_     LPCVOID                      pSrcData,
    _In_     SIZE_T                       SrcDataSize,
    _In_     D3DX11_IMAGE_LOAD_INFO      *pCreateInfo,
    _In_     ID3DX11ThreadPump          *pPump,
    _Out_    ID3D11ShaderResourceView **ppShaderResourceView,
    _Out_    HRESULT                     *pHResult
);
```

## Parameters

*pDevice* [in]

Type: [ID3D11Device](#)\*

A pointer to the device (see [ID3D11Device](#)) that will use the resource.

*pSrcData* [in]

Type: [LPCVOID](#)

Pointer to the file in memory that contains the shader-resource view.

*SrcDataSize* [in]

Type: [SIZE\\_T](#)

Size of the file in memory.

*pLoadInfo* [in]

Type: [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)\*

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*pPump* [in]

Type: [ID3DX11ThreadPump](#)\*

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). If **NULL** is specified, this function will behave synchronously and will not return until it is finished.

*ppShaderResourceView* [out]

Type: [ID3D11ShaderResourceView](#)\*\*

Address of a pointer to the newly created shader resource view. See [ID3D11ShaderResourceView](#).

*pHResult* [out]

Type: [HRESULT](#) ↴\*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateShaderResourceViewFromResource function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use **resource functions**, then these:

- [DirectXTK](#) library (runtime), [CreateXXXTextureFromMemory](#) (where XXX is DDS or WIC)
- [DirectXTex](#) library (tools), [LoadFromXXXMemory](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then [CreateShaderResourceView](#)

Create a shader-resource view from a resource.

## Syntax

C++

```
HRESULT D3DX11CreateShaderResourceViewFromResource(
    _In_     ID3D11Device                  *pDevice,
    _In_     HMODULE                      hSrcModule,
    _In_     LPCTSTR                       pSrcResource,
    _In_     D3DX11_IMAGE_LOAD_INFO        *pCreateInfo,
    _In_     ID3DX11ThreadPump            *pPump,
    _Out_    ID3D11ShaderResourceView **ppShaderResourceView,
    _Out_    HRESULT                      *pHResult
);
```

## Parameters

*pDevice* [in]

Type: [ID3D11Device\\*](#)

A pointer to the device (see [ID3D11Device](#)) that will use the resource.

*hSrcModule* [in]

Type: [HMODULE](#)

Handle to the resource module containing the shader-resource view. HMODULE can be obtained with [GetModuleHandle Function](#).

*pSrcResource* [in]

Type: [LPCTSTR](#)

Name of the shader resource view in *hSrcModule*. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*pLoadInfo* [in]

Type: [D3DX11\\_IMAGE\\_LOAD\\_INFO\\*](#)

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). If **NULL** is specified, this function will behave synchronously and will not return until it is finished.

*ppShaderResourceView* [out]

Type: [ID3D11ShaderResourceView\\*\\*](#)

Address of a pointer to the shader-resource view (see [ID3D11ShaderResourceView](#)).

*pHResult* [out]

Type: [HRESULT](#) ↴ \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

# Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateTextureFromFile function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use these:

- [DirectXTK](#) library (runtime), [CreateXXXTextureFromFile](#) (where XXX is DDS or WIC)
- [DirectXTex](#) library (tools), [LoadFromXXXFile](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then [CreateTexture](#)

Create a texture resource from a file.

## Syntax

C++

```
HRESULT D3DX11CreateTextureFromFile(
    _In_     ID3D11Device              *pDevice,
    _In_     LPCTSTR                  pSrcFile,
    _In_     D3DX11_IMAGE_LOAD_INFO   *pLoadInfo,
    _In_     ID3DX11ThreadPump       *pPump,
    _Out_    ID3D11Resource          **ppTexture,
    _Out_    HRESULT                 *pHResult
);
```

## Parameters

*pDevice* [in]

Type: [ID3D11Device\\*](#)

A pointer to the device (see [ID3D11Device](#)) that will use the resource.

*pSrcFile* [in]

Type: [LPCTSTR](#)

The name of the file containing the resource. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*pLoadInfo* [in]

Type: [D3DX11\\_IMAGE\\_LOAD\\_INFO\\*](#)

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). If **NULL** is specified, this function will behave synchronously and will not return until it is finished.

*ppTexture* [out]

Type: [ID3D11Resource\\*\\*](#)

The address of a pointer to the texture resource (see [ID3D11Resource](#)).

*pHResult* [out]

Type: [HRESULT](#) ↗ \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#) ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
-------------	-------

Requirement	Value
Header	D3DX11.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateTextureFromMemory function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use these:

- [DirectXTK](#) library (runtime), [CreateXXXTextureFromMemory](#) (where XXX is DDS or WIC)
- [DirectXTex](#) library (tools), [LoadFromXXXMemory](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then [CreateTexture](#)

Create a texture resource from a file residing in system memory.

## Syntax

C++

```
HRESULT D3DX11CreateTextureFromMemory(
    _In_ ID3D11Device              *pDevice,
    _In_ LPCVOID                   pSrcData,
    _In_ SIZE_T                    SrcDataSize,
    _In_ D3DX11_IMAGE_LOAD_INFO   *pCreateInfo,
    _In_ ID3DX11ThreadPump        *pPump,
    _Out_ ID3D11Resource          **ppTexture,
    _Out_ HRESULT                  *pHResult
);
```

## Parameters

*pDevice* [in]

Type: [ID3D11Device](#)\*

A pointer to the device (see [ID3D11Device](#)) that will use the resource.

*pSrcData* [in]

Type: [LPCVOID](#)

Pointer to the resource in system memory.

*SrcDataSize* [in]

Type: [SIZE\\_T](#)

Size of the resource in system memory.

*pLoadInfo* [in]

Type: [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)\*

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*pPump* [in]

Type: [ID3DX11ThreadPump](#)\*

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). If **NULL** is specified, this function will behave synchronously and will not return until it is finished.

*ppTexture* [out]

Type: [ID3D11Resource](#)\*\*

Address of a pointer to the created resource. See [ID3D11Resource](#).

*pHResult* [out]

Type: [HRESULT](#) ↗\*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#) ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateTextureFromResource function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use **resource functions**, then these:

- [DirectXTK](#) library (runtime), [CreateXXXTextureFromMemory](#) (where XXX is DDS or WIC)
- [DirectXToolKit](#) library (tools), [LoadFromXXXMemory](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games) then [CreateTexture](#)

Create a texture from another resource.

## Syntax

C++

```
HRESULT D3DX11CreateTextureFromResource(
    _In_     ID3D11Device              *pDevice,
    _In_     HMODULE                  hSrcModule,
    _In_     LPCTSTR                  pSrcResource,
    _In_     D3DX11_IMAGE_LOAD_INFO   *pCreateInfo,
    _In_     ID3DX11ThreadPump        *pPump,
    _Out_    ID3D11Resource           **ppTexture,
    _Out_    HRESULT                  *pHResult
);
```

## Parameters

*pDevice* [in]

Type: [ID3D11Device\\*](#)

A pointer to the device (see [ID3D11Device](#)) that will use the resource.

*hSrcModule* [in]

Type: [HMODULE](#)

A handle to the source resource. HMODULE can be obtained with [GetModuleHandle Function](#).

*pSrcResource* [in]

Type: [LPCTSTR](#)

A string that contains the name of the source resource. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*pLoadInfo* [in]

Type: [D3DX11\\_IMAGE\\_LOAD\\_INFO\\*](#)

Optional. Identifies the characteristics of a texture (see [D3DX11\\_IMAGE\\_LOAD\\_INFO](#)) when the data processor is created; set this to **NULL** to read the characteristics of a texture when the texture is loaded.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). If **NULL** is specified, this function will behave synchronously and will not return until it is finished.

*ppTexture* [out]

Type: [ID3D11Resource\\*\\*](#)

The address of a pointer to the texture resource (see [ID3D11Resource](#)).

*pHResult* [out]

Type: [HRESULT](#) ↴ \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

# Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11CreateThreadPump function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. See Remarks.

Create a thread pump.

## Syntax

C++

```
HRESULT D3DX11CreateThreadPump(
    _In_    UINT          cIoThreads,
    _In_    UINT          cProcThreads,
    _Out_   ID3DX11ThreadPump **ppThreadPump
);
```

## Parameters

*cIoThreads* [in]

Type: [UINT](#)

The number of I/O threads to create. If 0 is specified, Direct3D will attempt to calculate the optimal number of threads based on the hardware configuration.

*cProcThreads* [in]

Type: [UINT](#)

The number of process threads to create. If 0 is specified, Direct3D will attempt to calculate the optimal number of threads based on the hardware configuration.

*ppThreadPump* [out]

Type: [ID3DX11ThreadPump\\*\\*](#)

The created thread pump. See [ID3DX11ThreadPump Interface](#).

# Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

A thread pump is a very resource-intensive object. Only one thread pump should be created per application.

There s no implementation of the async loader outside of D3DX 10, and D3DX 11.

For Windows Store apps, the DirectX samples (for example, the [Direct3D tutorial sample](#)) include the **BasicLoader** module that uses the Windows Runtime asynchronous programming model ([AsyncBase](#)).

For Win32 desktop apps, you can use the [Concurrency Runtime](#) to implement something similar to the Windows Runtime asynchronous programming model.

## Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11FilterTexture function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [DirectXTex](#) library, [GenerateMipMaps](#) and [GenerateMipMaps3D](#).

Generates mipmap chain using a particular texture filter.

## Syntax

C++

```
HRESULT D3DX11FilterTexture(
    ID3D11DeviceContext *pContext,
    ID3D11Resource      *pTexture,
    UINT                SrcLevel,
    UINT                MipFilter
);
```

## Parameters

*pContext*

Type: [ID3D11DeviceContext\\*](#)

A pointer to an [ID3D11DeviceContext](#) object.

*pTexture*

Type: [ID3D11Resource\\*](#)

The texture object to be filtered. See [ID3D11Resource](#).

*SrcLevel*

Type: [UINT](#)

The mipmap level whose data is used to generate the rest of the mipmap chain.

*MipFilter*

Type: [UINT](#)

Flags controlling how each miplevel is filtered (or D3DX11\_DEFAULT for D3DX11\_FILTER\_LINEAR). See [D3DX11\\_FILTER\\_FLAG](#).

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11GetImageInfoFromFile function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [DirectXTex](#) library, `GetMetadataFromXXXFile` (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games).

Retrieves information about a given image file.

## Syntax

C++

```
HRESULT D3DX11GetImageInfoFromFile(
    _In_     LPCTSTR             pSrcFile,
    _In_     ID3DX11ThreadPump *pPump,
    _In_     D3DX11_IMAGE_INFO *pSrcInfo,
    _Out_    HRESULT             *pHResult
);
```

## Parameters

*pSrcFile* [in]

Type: [LPCTSTR](#)

File name of image to retrieve information about. If UNICODE or \_UNICODE are defined, this parameter type is LPCWSTR, otherwise, the type is LPCSTR.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

Optional thread pump that can be used to load the info asynchronously. Can be **NULL**. See [ID3DX11ThreadPump Interface](#).

*pSrcInfo* [in]

Type: [D3DX11\\_IMAGE\\_INFO](#)\*

Pointer to a [D3DX11\\_IMAGE\\_INFO](#) to be filled with the description of the data in the source file.

*pHResult* [out]

Type: [HRESULT](#) \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#)

If the function succeeds, the return value is **D3D\_OK**. If the function fails, the return value can be the following: **D3DERR\_INVALIDCALL**

## Remarks

This function supports both Unicode and ANSI strings.

## Requirements

Requirement	Value
Header	<a href="#">D3DX11tex.h</a>
Library	<a href="#">D3DX11.lib</a>

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3DX11GetImageInfoFromMemory function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [DirectXTex](#) library, [GetMetadataFromXXXMemory](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games).

Get information about an image already loaded into memory.

## Syntax

C++

```
HRESULT D3DX11GetImageInfoFromMemory(
    _In_    LPCVOID           pSrcData,
    _In_    SIZE_T            SrcDataSize,
    _In_    ID3DX11ThreadPump *pPump,
    _In_    D3DX11_IMAGE_INFO *pSrcInfo,
    _Out_   HRESULT           *pHResult
);
```

## Parameters

*pSrcData* [in]

Type: [LPCVOID](#)

Pointer to the image in memory.

*SrcDataSize* [in]

Type: [SIZE\\_T](#)

Size of the image in memory, in bytes.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

Optional thread pump that can be used to load the info asynchronously. Can be **NULL**. See [ID3DX11ThreadPump Interface](#).

*pSrcInfo* [in]

Type: [D3DX11\\_IMAGE\\_INFO\\*](#)

Information about the image in memory.

*pHResult* [out]

Type: [HRESULT](#) \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3DX11GetImageInfoFromResource function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use [resource functions](#), then use [DirectXTex](#) library (tools), **LoadFromXXXMemory** (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games).

Retrieves information about a given image in a resource.

## Syntax

C++

```
HRESULT D3DX11GetImageInfoFromResource(
    _In_     HMODULE           hSrcModule,
    _In_     LPCTSTR            pSrcResource,
    _In_     ID3DX11ThreadPump *pPump,
    _In_     D3DX11_IMAGE_INFO *pSrcInfo,
    _Out_    HRESULT            *pHResult
);
```

## Parameters

*hSrcModule* [in]

Type: [HMODULE](#)

Module where the resource is loaded. Set this parameter to **NULL** to specify the module associated with the image that the operating system used to create the current process.

*pSrcResource* [in]

Type: [LPCTSTR](#)

Pointer to a string that specifies the filename. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR. See Remarks.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

Optional thread pump that can be used to load the info asynchronously. Can be **NULL**. See [ID3DX11ThreadPump Interface](#).

*pSrcInfo* [in]

Type: [D3DX11\\_IMAGE\\_INFO\\*](#)

Pointer to a D3DX11\_IMAGE\_INFO structure to be filled with the description of the data in the source file.

*pHResult* [out]

Type: [HRESULT](#) ↗\*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#) ↗

If the function succeeds, the return value is D3D\_OK. If the function fails, the return value can be the following: D3DERR\_INVALIDCALL

## Remarks

The compiler setting also determines the function version. If Unicode is defined, the function call resolves to **D3DX11GetImageInfoFromResourceW**. Otherwise, the function call resolves to **D3DX11GetImageInfoFromResourceA** because ANSI strings are being used.

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11LoadTextureFromTexture function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [DirectXTex](#) library, **Resize**, **Convert**, **Compress**, **Decompress**, and/or **CopyRectangle**.

Load a texture from a texture.

## Syntax

C++

```
HRESULT D3DX11LoadTextureFromTexture(
    ID3D11DeviceContext      *pContext,
    ID3D11Resource           *pSrcTexture,
    D3DX11_TEXTURE_LOAD_INFO *pCreateInfo,
    ID3D11Resource           *pDstTexture
);
```

## Parameters

*pContext*

Type: [ID3D11DeviceContext\\*](#)

A pointer to an [ID3D11DeviceContext](#) object.

*pSrcTexture*

Type: [ID3D11Resource\\*](#)

Pointer to the source texture. See [ID3D11Resource](#).

*pLoadInfo*

Type: [D3DX11\\_TEXTURE\\_LOAD\\_INFO\\*](#)

Pointer to texture loading parameters. See [D3DX11\\_TEXTURE\\_LOAD\\_INFO](#).

*pDstTexture*

Type: [ID3D11Resource\\*](#)

Pointer to the destination texture. See [ID3D11Resource](#).

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11PreprocessShaderFromFile function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [D3DPreprocess API](#).

Create a shader from a file without compiling it.

## Syntax

C++

```
HRESULT D3DX11PreprocessShaderFromFile(  
    _In_          LPCTSTR           pFileName,  
    _In_  const D3D11_SHADER_MACRO *pDefines,  
    _In_          LPD3D10INCLUDE    pInclude,  
    _In_          ID3DX11ThreadPump *pPump,  
    _Out_         ID3D10Blob       **ppShaderText,  
    _Out_         ID3D10Blob       **ppErrorMsgs,  
    _Out_         HRESULT           *pHResult  
) ;
```

## Parameters

*pFileName* [in]

Type: [LPCTSTR](#)

Name of the shader file.

*pDefines* [in]

Type: [const D3D11\\_SHADER\\_MACRO\\*](#)

A NULL-terminated array of shader macros; set this to **NULL** to specify no macros.

*pInclude* [in]

Type: [LPD3D10INCLUDE](#)

A pointer to an include interface; set this to **NULL** to specify there is no include file.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). Use **NULL** to specify that this function should not return until it is completed.

*ppShaderText* [out]

Type: [ID3D10Blob\\*\\*](#)

A pointer to memory that contains the uncompiled shader.

*ppErrorMsgs* [out]

Type: [ID3D10Blob\\*\\*](#)

The address of a pointer to memory that contains effect-creation errors, if any occurred.

*pHResult* [out]

Type: [HRESULT](#) \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11async.h

Requirement	Value
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11PreprocessShaderFromMemory function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [D3DPreprocess API](#).

Create a shader from memory without compiling it.

## Syntax

C++

```
HRESULT D3DX11PreprocessShaderFromMemory(
    _In_          LPCSTR           pSrcData,
    _In_          SIZE_T            SrcDataSize,
    _In_          LPCSTR           pFileName,
    _In_  const D3D11_SHADER_MACRO *pDefines,
    _In_          LPD3D10INCLUDE   pInclude,
    _In_          ID3DX11ThreadPump *pPump,
    _Out_         ID3D10Blob       **ppShaderText,
    _Out_         ID3D10Blob       **ppErrorMsgs,
    _Out_         HRESULT          *pHResult
);
```

## Parameters

*pSrcData* [in]

Type: [LPCSTR](#)

Pointer to the memory containing the shader.

*SrcDataSize* [in]

Type: [SIZE\\_T](#)

Size of the shader.

*pFileName* [in]

Type: [LPCSTR](#)

Name of the shader.

*pDefines* [in]

Type: [const D3D11\\_SHADER\\_MACRO\\*](#)

A NULL-terminated array of shader macros; set this to **NULL** to specify no macros.

*pInclude* [in]

Type: [LPD3D10INCLUDE](#)

A pointer to an include interface; set this to **NULL** to specify there is no include file.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). Use **NULL** to specify that this function should not return until it is completed.

*ppShaderText* [out]

Type: [ID3D10Blob\\*\\*](#)

A pointer to memory that contains the uncompiled shader.

*ppErrorMsgs* [out]

Type: [ID3D10Blob\\*\\*](#)

The address of a pointer to memory that contains effect-creation errors, if any occurred.

*pHResult* [out]

Type: [HRESULT](#) ↴ \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

# Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11PreprocessShaderFromResource function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [D3DPreprocess API](#).

Create a shader from a resource without compiling it.

## Syntax

C++

```
HRESULT D3DX11PreprocessShaderFromResource(
    _In_          HMODULE           hModule,
    _In_          LPCTSTR           pResourceName,
    _In_          LPCTSTR           pSrcFileName,
    _In_  const D3D11_SHADER_MACRO *pDefines,
    _In_          LPD3D10INCLUDE    pInclude,
    _In_          ID3DX11ThreadPump *pPump,
    _Out_         ID3D10Blob       **ppShaderText,
    _Out_         ID3D10Blob       **ppErrorMsgs,
    _Out_         HRESULT           *pHResult
);
```

## Parameters

*hModule* [in]

Type: [HMODULE](#)

Handle to the resource module containing the shader. HMODULE can be obtained with [GetModuleHandle Function](#).

*pResourceName* [in]

Type: [LPCTSTR](#)

The name of the resource in side hModule containing the shader. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

*pSrcFileName* [in]

Type: [LPCTSTR](#)

Optional. Effect file name, which is used for error messages only. Can be **NULL**.

*pDefines* [in]

Type: [const D3D11\\_SHADER\\_MACRO\\*](#)

A NULL-terminated array of shader macros; set this to **NULL** to specify no macros.

*pInclude* [in]

Type: [LPD3D10INCLUDE](#)

A pointer to an include interface; set this to **NULL** to specify there is no include file.

*pPump* [in]

Type: [ID3DX11ThreadPump\\*](#)

A pointer to a thread pump interface (see [ID3DX11ThreadPump Interface](#)). Use **NULL** to specify that this function should not return until it is completed.

*ppShaderText* [out]

Type: [ID3D10Blob\\*\\*](#)

A pointer to memory that contains the uncompiled shader.

*ppErrorMsgs* [out]

Type: [ID3D10Blob\\*\\*](#)

The address of a pointer to memory that contains effect-creation errors, if any occurred.

*pHResult* [out]

Type: [HRESULT](#) ↗ \*

A pointer to the return value. May be **NULL**. If *pPump* is not **NULL**, then *pHResult* must be a valid memory location until the asynchronous execution completes.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11async.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11SaveTextureToFile function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [DirectXTex](#) library, [CaptureTexture](#) then [SaveToXXXFile](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games). For the simplified scenario of creating a screen shot from a render target texture, we recommend that you use the [DirectXTK](#) library, [SaveDDSTextureToFile](#) or [SaveWICTextureToFile](#).

Save a texture to a file.

## Syntax

C++

```
HRESULT D3DX11SaveTextureToFile(
    ID3D11DeviceContext      *pContext,
    _In_ ID3D11Resource       *pSrcTexture,
    _In_ D3DX11_IMAGE_FILE_FORMAT DestFormat,
    _In_ LPCTSTR              pDestFile
);
```

## Parameters

*pContext*

Type: [ID3D11DeviceContext\\*](#)

A pointer to an [ID3D11DeviceContext](#) object.

*pSrcTexture* [in]

Type: [ID3D11Resource](#)\*

Pointer to the texture to be saved. See [ID3D11Resource](#).

*DestFormat* [in]

Type: [D3DX11\\_IMAGE\\_FILE\\_FORMAT](#)

The format the texture will be saved as (see [D3DX11\\_IMAGE\\_FILE\\_FORMAT](#)).

D3DX11\_IFF\_DDS is the preferred format since it is the only option that supports all the formats in [DXGI\\_FORMAT](#).

*pDestFile* [in]

Type: [LPCTSTR](#)

Name of the destination output file where the texture will be saved. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the data type resolves to LPCSTR.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#); use the return value to see if the *DestFormat* is supported.

## Remarks

`D3DX11SaveTextureToFile` writes out the extra [DDS\\_HEADER\\_DXT10](#) structure for the input texture only if necessary (for example, because the input texture is in standard RGB (sRGB) format). If `D3DX11SaveTextureToFile` writes out the [DDS\\_HEADER\\_DXT10](#) structure, it sets the `dwFourCC` member of the [DDS\\_PIXELFORMAT](#) structure for the texture to [DX10](#) to indicate the presence of the [DDS\\_HEADER\\_DXT10](#) extended header.

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11SaveTextureToMemory function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [DirectXTex](#) library, [CaptureTexture](#) then [SaveToXXXMemory](#) (where XXX is WIC, DDS, or TGA; WIC doesn't support DDS and TGA; D3DX 9 supported TGA as a common art source format for games).

Save a texture to memory.

## Syntax

C++

```
HRESULT D3DX11SaveTextureToMemory(
    ID3D11DeviceContext      *pContext,
    _In_   ID3D11Resource      *pSrcTexture,
    _In_   D3DX11_IMAGE_FILE_FORMAT DestFormat,
    _Out_  LPD3D10BLOB        *ppDestBuf,
    _In_   UINT                 Flags
);
```

## Parameters

*pContext*

Type: [ID3D11DeviceContext\\*](#)

A pointer to an [ID3D11DeviceContext](#) object.

*pSrcTexture* [in]

Type: [ID3D11Resource\\*](#)

Pointer to the texture to be saved. See [ID3D11Resource](#).

*DestFormat* [in]

Type: [D3DX11\\_IMAGE\\_FILE\\_FORMAT](#)

The format the texture will be saved as. See [D3DX11\\_IMAGE\\_FILE\\_FORMAT](#).

*ppDestBuf* [out]

Type: [LPD3D10BLOB\\*](#)

Address of a pointer to the memory containing the saved texture.

*Flags* [in]

Type: [UINT](#)

Optional.

## Return value

Type: [HRESULT](#) ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3DX11SHProjectCubeMap function

Article • 04/28/2022

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [Spherical Harmonics Math](#) library function [SHProjectCubeMap](#).

Projects a function represented in a cube map into spherical harmonics.

## Syntax

C++

```
HRESULT D3DX11SHProjectCubeMap(
    ID3D11DeviceContext *pContext,
    UINT                Order,
    ID3D11Texture2D     *pCubeMap,
    FLOAT               *pROut,
    FLOAT               *pGOut,
    FLOAT               *pBOut
);
```

## Parameters

*pContext*

Type: [ID3D11DeviceContext\\*](#)

A pointer to an [ID3D11DeviceContext](#) object.

*Order*

Type: [UINT](#)

Order of the SH evaluation, generates Order<sup>2</sup> coefficients whose degree is Order-1.  
Valid range is between 2 and 6.

*pCubeMap*

Type: [ID3D11Texture2D\\*](#)

A pointer to an [ID3D11Texture2D](#) that represents a cubemap that is going to be projected into spherical harmonics.

*pROut*

Type: [FLOAT\\*](#)

Output SH vector for red.

*pGOut*

Type: [FLOAT\\*](#)

Output SH vector for green.

*pBOut*

Type: [FLOAT\\*](#)

Output SH vector for blue.

## Return value

Type: [HRESULT](#) ↗

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11UnsetAllDeviceObjects function

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## ⓘ Note

Instead of using this function, we recommend that you use the [ID3D11DeviceContext::ClearState](#) method.

Removes all resources from the device by setting their pointers to **NULL**. This should be called during shutdown of your application. It helps ensure that when one is releasing all of their resources that none of them are bound to the device.

## Syntax

C++

```
HRESULT D3DX11UnsetAllDeviceObjects(
    _In_ ID3D11DeviceContext *pContext
);
```

## Parameters

*pContext* [in]

Type: [ID3D11DeviceContext\\*](#)

Pointer to an [ID3D11DeviceContext](#) object.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

# Requirements

Requirement	Value
Header	D3DX11core.h
Library	D3DX11.lib

## See also

[D3DX Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX Structures (Direct3D 11 Graphics)

Article • 08/18/2023

This section contains information about the D3DX 11 structures.

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## In this section

Topic	Description
<a href="#">D3DX11_IMAGE_INFO</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Optionally provide information to texture loader APIs to control how textures get loaded. A value of D3DX11_DEFAULT for any of these parameters will cause D3DX to automatically use the value from the source file.</p>
<a href="#">D3DX11_IMAGE_LOAD_INFO</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Optionally provide information to texture loader APIs to control how textures get loaded. A value of D3DX11_DEFAULT for any of these parameters will cause D3DX to automatically use the value from the source file.</p>
<a href="#">D3DX11_TEXTURE_LOAD_INFO</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Describes parameters used to load a texture from another texture.</p>

## Related topics

[D3DX 11 Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3DX11\_IMAGE\_INFO structure

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Optionally provide information to texture loader APIs to control how textures get loaded. A value of D3DX11\_DEFAULT for any of these parameters will cause D3DX to automatically use the value from the source file.

## Syntax

C++

```
typedef struct D3DX11_IMAGE_INFO {
    UINT             Width;
    UINT             Height;
    UINT             Depth;
    UINT             ArraySize;
    UINT             MipLevels;
    UINT             MiscFlags;
    DXGI_FORMAT      Format;
    D3D11_RESOURCE_DIMENSION ResourceDimension;
    D3DX11_IMAGE_FILE_FORMAT ImageFileFormat;
} D3DX11_IMAGE_INFO, *LPD3DX11_IMAGE_INFO;
```

## Members

### Width

Type: [UINT](#)

The target width of the texture. If the actual width of the texture is larger or smaller than this value then the texture will be scaled up or down to fit this target width.

### Height

Type: [UINT](#)

The target height of the texture. If the actual height of the texture is larger or smaller than this value then the texture will be scaled up or down to fit this target height.

### Depth

Type: [UINT](#)

The depth of the texture. This only applies to volume textures.

### ArraySize

Type: [UINT](#)

The number of elements in the array.

### MipLevels

Type: [UINT](#)

The maximum number of mipmap levels in the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#). Using 0 or D3DX11\_DEFAULT will cause a full mipmap chain to be created.

### MiscFlags

Type: [UINT](#)

Miscellaneous resource properties specified with a [D3D11\\_RESOURCE\\_MISC\\_FLAG](#) flag.

### Format

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#) enumeration specifying the format the texture will be in after it is loaded.

### ResourceDimension

Type: [D3D11\\_RESOURCE\\_DIMENSION](#)

A [D3D11\\_RESOURCE\\_DIMENSION](#) value, which identifies the type of resource.

### ImageFileFormat

Type: [D3DX11\\_IMAGE\\_FILE\\_FORMAT](#)

A [D3DX11\\_IMAGE\\_FILE\\_FORMAT](#) value, which identifies the image format.

## Remarks

This structure is used by methods such as: [D3DX11GetImageInfoFromFile](#), [D3DX11GetImageInfoFromMemory](#), or [D3DX11GetImageInfoFromResource](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h

## See also

[D3DX Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_IMAGE\_LOAD\_INFO structure

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Optionally provide information to texture loader APIs to control how textures get loaded. A value of D3DX11\_DEFAULT for any of these parameters will cause D3DX to automatically use the value from the source file.

## Syntax

C++

```
typedef struct D3DX11_IMAGE_LOAD_INFO {
    UINT           Width;
    UINT           Height;
    UINT           Depth;
    UINT           FirstMipLevel;
    UINT           MipLevels;
    D3D11_USAGE   Usage;
    UINT           BindFlags;
    UINT           CpuAccessFlags;
    UINT           MiscFlags;
    DXGI_FORMAT    Format;
    UINT           Filter;
    UINT           MipFilter;
    D3DX11_IMAGE_INFO *pSrcInfo;
} D3DX11_IMAGE_LOAD_INFO, *LPD3DX11_IMAGE_LOAD_INFO;
```

## Members

### Width

Type: [UINT](#)

The target width of the texture. If the actual width of the texture is larger or smaller than this value then the texture will be scaled up or down to fit this target width.

### Height

Type: [UINT](#)

The target height of the texture. If the actual height of the texture is larger or smaller than this value then the texture will be scaled up or down to fit this target height.

### Depth

Type: [UINT](#)

The depth of the texture. This only applies to volume textures.

### FirstMipLevel

Type: [UINT](#)

The highest resolution mipmap level of the texture. If this is greater than 0, then after the texture is loaded FirstMipLevel will be mapped to mipmap level 0.

### MipLevels

Type: [UINT](#)

The maximum number of mipmap levels in the texture. See the remarks in [D3D11\\_TEX1D\\_SRV](#). Using 0 or D3DX11\_DEFAULT will cause a full mipmap chain to be created.

### Usage

Type: [D3D11\\_USAGE](#)

The way the texture resource is intended to be used. See [D3D11\\_USAGE](#).

### BindFlags

Type: [UINT](#)

The pipeline stages that the texture will be allowed to bind to. See [D3D11\\_BIND\\_FLAG](#).

### CpuAccessFlags

Type: [UINT](#)

The access permissions the cpu will have for the texture resource. See [D3D11\\_CPU\\_ACCESS\\_FLAG](#).

### MiscFlags

Type: [UINT](#)

Miscellaneous resource properties (see [D3D11\\_RESOURCE\\_MISC\\_FLAG](#)).

## Format

Type: [DXGI\\_FORMAT](#)

A [DXGI\\_FORMAT](#) enumeration indicating the format the texture will be in after it is loaded.

## Filter

Type: [UINT](#)

Filter the texture using the specified filter (only when resampling). See [D3DX11\\_FILTER\\_FLAG](#).

## MipFilter

Type: [UINT](#)

Filter the texture mip levels using the specified filter (only if generating mipmaps). Valid values are D3DX11\_FILTER\_NONE, D3DX11\_FILTER\_POINT, D3DX11\_FILTER\_LINEAR, or D3DX11\_FILTER\_TRIANGLE. See [D3DX11\\_FILTER\\_FLAG](#).

## pSrcInfo

Type: [D3DX11\\_IMAGE\\_INFO\\*](#)

Information about the original image. See [D3DX11\\_IMAGE\\_INFO](#). Can be obtained with [D3DX11GetImageInfoFromFile](#), [D3DX11GetImageInfoFromMemory](#), or [D3DX11GetImageInfoFromResource](#).

## Remarks

When initializing the structure, you may set any member to D3DX11\_DEFAULT and D3DX will initialize it with a default value from the source texture when the texture is loaded.

This structure can be used by APIs that:

- Create resources, such as [D3DX11CreateTextureFromFile](#) and [D3DX11CreateShaderResourceViewFromFile](#).
- Create data processors, such as [D3DX11CreateAsyncTextureInfoProcessor](#) or [D3DX11CreateAsyncShaderResourceViewProcessor](#).

The default values are:

```
Width = D3DX11_DEFAULT;
Height = D3DX11_DEFAULT;
Depth = D3DX11_DEFAULT;
FirstMipLevel = D3DX11_DEFAULT;
MipLevels = D3DX11_DEFAULT;
Usage = (D3D11_USAGE) D3DX11_DEFAULT;
BindFlags = D3DX11_DEFAULT;
CpuAccessFlags = D3DX11_DEFAULT;
MiscFlags = D3DX11_DEFAULT;
Format = DXGI_FORMAT_FROM_FILE;
Filter = D3DX11_DEFAULT;
MipFilter = D3DX11_DEFAULT;
pSrcInfo = NULL;
```

Here is a brief example that uses this structure to supply the pixel format when loading a texture. For the complete code, see [HDRFormats10.cpp](#) in [HDRToneMappingCS11 Sample](#).

```
ID3D11ShaderResourceView* pCubeRV = NULL;
WCHAR strPath[MAX_PATH];
D3DX11_IMAGE_LOAD_INFO LoadInfo;

DXUTFindDXSDKMediaFileCch( strPath, MAX_PATH,
    L"Light Probes\\uffizi_cross.dds" );

LoadInfo.Format = DXGI_FORMAT_R16G16B16A16_FLOAT;

hr = D3DX11CreateShaderResourceViewFromFile( pd3dDevice, strPath,
    &LoadInfo, NULL, &pCubeRV, NULL );
```

## Requirements

Requirement	Value
Header	D3DX11tex.h

## See also

[D3DX Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# D3DX11\_TEXTURE\_LOAD\_INFO structure

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Describes parameters used to load a texture from another texture.

## Syntax

C++

```
typedef struct _D3DX11_TEXTURE_LOAD_INFO {
    D3D11_BOX *pSrcBox;
    D3D11_BOX *pDstBox;
    UINT        SrcFirstMip;
    UINT        DstFirstMip;
    UINT        NumMips;
    UINT        SrcFirstElement;
    UINT        DstFirstElement;
    UINT        NumElements;
    UINT        Filter;
    UINT        MipFilter;
} D3DX11_TEXTURE_LOAD_INFO;
```

## Members

### pSrcBox

Type: [D3D11\\_BOX\\*](#)

Source texture box (see [D3D11\\_BOX](#)).

### pDstBox

Type: [D3D11\\_BOX\\*](#)

Destination texture box (see [D3D11\\_BOX](#)).

### SrcFirstMip

Type: [UINT](#)

Source texture mipmap level, see [D3D11CalcSubresource](#) for more detail.

### DstFirstMip

Type: [UINT](#)

Destination texture mipmap level, see [D3D11CalcSubresource](#) for more detail.

### NumMips

Type: [UINT](#)

Number of mipmap levels in the source texture.

### SrcFirstElement

Type: [UINT](#)

First element of the source texture.

### DstFirstElement

Type: [UINT](#)

First element of the destination texture.

### NumElements

Type: [UINT](#)

Number of elements to load.

### Filter

Type: [UINT](#)

Filtering options during resampling (see [D3DX11\\_FILTER\\_FLAG](#)).

### MipFilter

Type: [UINT](#)

Filtering options when generating mip levels (see [D3DX11\\_FILTER\\_FLAG](#)).

## Remarks

This structure is used in a call to [D3DX11LoadTextureFromTexture](#).

The default values are:

```
pSrcBox = NULL;  
pDstBox = NULL;  
SrcFirstMip = 0;  
DstFirstMip = 0;  
NumMips = D3DX11_DEFAULT;  
SrcFirstElement = 0;  
DstFirstElement = 0;  
NumElements = D3DX11_DEFAULT;  
Filter = D3DX11_DEFAULT;  
MipFilter = D3DX11_DEFAULT;
```

## Requirements

Requirement	Value
Header	D3DX11tex.h

## See also

[D3DX Structures](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX Enumerations (Direct3D 11 Graphics)

Article • 08/18/2023

This section contains information about D3DX 11 enumerations.

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

## In this section

Topic	Description
<a href="#">D3DX11_CHANNEL_FLAG</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>These flags are used by functions which operate on one or more channels in a texture.</p>
<a href="#">D3DX11_ERR</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Errors are represented by negative values and cannot be combined. The following is a list of values that can be returned by methods included with the D3DX utility library. See the individual method descriptions for lists of the values that each can return. These lists are not necessarily comprehensive.</p>
<a href="#">D3DX11_FILTER_FLAG</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Texture filtering flags.</p>
<a href="#">D3DX11_IMAGE_FILE_FORMAT</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Image file formats supported by D3DX11CreateXXX and D3DX11SaveXXX functions.</p>
<a href="#">D3DX11_NORMALMAP_FLAG</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p>

Topic	Description
	Normal map options. You can combine any number of these flags by using a bitwise OR operation.
<a href="#">D3DX11_SAVE_TEXTURE_FLAG</a>	<p><b>Note:</b> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.</p> <p>Texture save options.</p>

## Related topics

[D3DX 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_CHANNEL\_FLAG enumeration

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

These flags are used by functions which operate on one or more channels in a texture.

## Syntax

C++

```
typedef enum D3DX11_CHANNEL_FLAG {
    D3DX11_CHANNEL_RED      = (1 << 0),
    D3DX11_CHANNEL_BLUE     = (1 << 1),
    D3DX11_CHANNEL_GREEN    = (1 << 2),
    D3DX11_CHANNEL_ALPHA    = (1 << 3),
    D3DX11_CHANNEL_LUMINANCE = (1 << 4)
} D3DX11_CHANNEL_FLAG, *LPD3DX11_CHANNEL_FLAG;
```

## Constants

### D3DX11\_CHANNEL\_RED

Indicates the red channel should be used.

### D3DX11\_CHANNEL\_BLUE

Indicates the blue channel should be used.

### D3DX11\_CHANNEL\_GREEN

Indicates the green channel should be used.

### D3DX11\_CHANNEL\_ALPHA

Indicates the alpha channel should be used.

### D3DX11\_CHANNEL\_LUMINANCE

Indicates the luminances of the red, green, and blue channels should be used.

# Requirements

Requirement	Value
Header	D3DX11tex.h

## See also

[D3DX Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_ERR enumeration

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Errors are represented by negative values and cannot be combined. The following is a list of values that can be returned by methods included with the D3DX utility library. See the individual method descriptions for lists of the values that each can return. These lists are not necessarily comprehensive.

## Syntax

C++

```
typedef enum D3DX11_ERR {
    D3DX11_ERR_CANNOT_MODIFY_INDEX_BUFFER = MAKE_DDHRESULT(2900),
    D3DX11_ERR_INVALID_MESH               = MAKE_DDHRESULT(2901),
    D3DX11_ERR_CANNOT_ATTR_SORT          = MAKE_DDHRESULT(2902),
    D3DX11_ERR_SKINNING_NOT_SUPPORTED    = MAKE_DDHRESULT(2903),
    D3DX11_ERR_TOO_MANY_INFLUENCES      = MAKE_DDHRESULT(2904),
    D3DX11_ERR_INVALID_DATA              = MAKE_DDHRESULT(2905),
    D3DX11_ERR_LOADED_MESH_HAS_NO_DATA   = MAKE_DDHRESULT(2906),
    D3DX11_ERR_DUPLICATE_NAMED_FRAGMENT = MAKE_DDHRESULT(2907),
    D3DX11_ERR_CANNOT_REMOVE_LAST_ITEM   = MAKE_DDHRESULT(2908)
} D3DX11_ERR, *LPD3DX11_ERR;
```

## Constants

### D3DX11\_ERR\_CANNOT\_MODIFY\_INDEX\_BUFFER

The index buffer cannot be modified.

### D3DX11\_ERR\_INVALID\_MESH

The mesh is invalid.

### D3DX11\_ERR\_CANNOT\_ATTR\_SORT

Attribute sort (D3DXMESHOPT\_ATTRSORT) is not supported as an optimization technique.

### D3DX11\_ERR\_SKINNING\_NOT\_SUPPORTED

Skinning is not supported.

### D3DX11\_ERR\_TOO\_MANY\_INFLUENCES

Too many influences specified.

### D3DX11\_ERR\_INVALID\_DATA

The data is invalid.

### D3DX11\_ERR\_LOADED\_MESH\_HAS\_NO\_DATA

The mesh has no data.

### D3DX11\_ERR\_DUPLICATE\_NAMED\_FRAGMENT

A fragment with that name already exists.

### D3DX11\_ERR\_CANNOT\_REMOVE\_LAST\_ITEM

The last item cannot be deleted.

## Remarks

The facility code \_FACDD is used to generate error codes, as in the following macros.

```
#define _FACDD          0x876
#define MAKE_DDHRESULT( code )  MAKE_HRESULT( 1, _FACDD, code )
enum _D3DXERR {
    D3DXERR_CANNOTMODIFYINDEXBUFFER = MAKE_DDHRESULT(2900),
    D3DXERR_INVALIDDMESH           = MAKE_DDHRESULT(2901),
    ...
};
```

## Requirements

Requirement	Value
Header	D3DX11.h

## See also

[D3DX Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_FILTER\_FLAG enumeration

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Texture filtering flags.

## Syntax

C++

```
typedef enum D3DX11_FILTER_FLAG {
    D3DX11_FILTER_NONE          = (1 << 0),
    D3DX11_FILTER_POINT         = (2 << 0),
    D3DX11_FILTER_LINEAR        = (3 << 0),
    D3DX11_FILTER_TRIANGLE      = (4 << 0),
    D3DX11_FILTER_BOX           = (5 << 0),
    D3DX11_FILTER_MIRROR_U      = (1 << 16),
    D3DX11_FILTER_MIRROR_V      = (2 << 16),
    D3DX11_FILTER_MIRROR_W      = (4 << 16),
    D3DX11_FILTER_MIRROR       = (7 << 16),
    D3DX11_FILTER_DITHER        = (1 << 19),
    D3DX11_FILTER_DITHER_DIFFUSION = (2 << 19),
    D3DX11_FILTER_SRGB_IN       = (1 << 21),
    D3DX11_FILTER_SRGB_OUT      = (2 << 21),
    D3DX11_FILTER_SRGB          = (3 << 21)
} D3DX11_FILTER_FLAG, *LPD3DX11_FILTER_FLAG;
```

## Constants

### D3DX11\_FILTER\_NONE

No scaling or filtering will take place. Pixels outside the bounds of the source image are assumed to be transparent black.

### D3DX11\_FILTER\_POINT

Each destination pixel is computed by sampling the nearest pixel from the source image.

### D3DX11\_FILTER\_LINEAR

Each destination pixel is computed by sampling the four nearest pixels from the source image. This filter works best when the scale on both axes is less than two.

### **D3DX11\_FILTER\_TRIANGLE**

Every pixel in the source image contributes equally to the destination image. This is the slowest of the filters.

### **D3DX11\_FILTER\_BOX**

Each pixel is computed by averaging a 2x2(x2) box of pixels from the source image. This filter works only when the dimensions of the destination are half those of the source, as is the case with mipmaps.

### **D3DX11\_FILTER\_MIRROR\_U**

Pixels off the edge of the texture on the u-axis should be mirrored, not wrapped.

### **D3DX11\_FILTER\_MIRROR\_V**

Pixels off the edge of the texture on the v-axis should be mirrored, not wrapped.

### **D3DX11\_FILTER\_MIRROR\_W**

Pixels off the edge of the texture on the w-axis should be mirrored, not wrapped.

### **D3DX11\_FILTER\_MIRROR**

Specifying this flag is the same as specifying the D3DX\_FILTER\_MIRROR\_U, D3DX\_FILTER\_MIRROR\_V, and D3DX\_FILTER\_MIRROR\_W flags.

### **D3DX11\_FILTER\_DITHER**

The resulting image must be dithered using a 4x4 ordered dither algorithm. This happens when converting from one format to another.

### **D3DX11\_FILTER\_DITHER\_DIFFUSION**

Do diffuse dithering on the image when changing from one format to another.

### **D3DX11\_FILTER\_SRGB\_IN**

Input data is in standard RGB (sRGB) color space. See remarks.

### **D3DX11\_FILTER\_SRGB\_OUT**

Output data is in standard RGB (sRGB) color space. See remarks.

## D3DX11\_FILTER\_SRGB

Same as specifying D3DX\_FILTER\_SRGB\_IN | D3DX\_FILTER\_SRGB\_OUT. See remarks.

## Remarks

D3DX11 automatically performs gamma correction (to convert color data from RGB space to standard RGB space) when loading texture data. This is automatically done for instance when RGB data is loaded from a .png file into an sRGB texture. Use the SRGB filter flags to indicate if the data does not need to be converted into sRGB space.

## Requirements

Requirement	Value
Header	D3DX11tex.h

## See also

[D3DX Enumerations](#)

---

## Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

# D3DX11\_IMAGE\_FILE\_FORMAT enumeration

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Image file formats supported by D3DX11Createxxx and D3DX11Savexxx functions.

## Syntax

C++

```
typedef enum D3DX11_IMAGE_FILE_FORMAT {
    D3DX11_IFF_BMP          = 0,
    D3DX11_IFF_JPG          = 1,
    D3DX11_IFF_PNG          = 3,
    D3DX11_IFF_DDS          = 4,
    D3DX11_IFF_TIFF         = 10,
    D3DX11_IFF_GIF          = 11,
    D3DX11_IFF_WMP          = 12,
    D3DX11_IFF_FORCE_DWORD   = 0x7fffffff
} D3DX11_IMAGE_FILE_FORMAT, *LPD3DX11_IMAGE_FILE_FORMAT;
```

## Constants

### D3DX11\_IFF\_BMP

Windows bitmap (BMP) file format. Contains a header that describes the resolution of the device on which the rectangle of pixels was created, the dimensions of the rectangle, the size of the array of bits, a logical palette, and an array of bits that defines the relationship between pixels in the bitmapped image and entries in the logical palette. The file extension for this format is .bmp.

### D3DX11\_IFF\_JPG

Joint Photographic Experts Group (JPEG) compressed file format. Specifies variable compression of 24-bit RGB color and 8-bit gray-scale Tagged Image File Format (TIFF) image document files. The file extension for this format is .jpg.

## D3DX11IFF\_PNG

Portable Network Graphics (PNG) file format. A non-proprietary bitmap format using lossless compression. The file extension for this format is .png.

## D3DX11IFF\_DDS

DirectDraw surface (DDS) file format. Stores textures, volume textures, and cubic environment maps, with or without mipmap levels, and with or without pixel compression. The file extension for this format is .dds.

## D3DX11IFF\_TIFF

Tagged Image File Format (TIFF). The file extensions for this format are .tif and .tiff.

## D3DX11IFF\_GIF

Graphics Interchange Format (GIF). The file extension for this format is .gif.

## D3DX11IFF\_WMP

Windows Media Photo format (WMP). This format is also known as HD Photo and JPEG XR. The file extensions for this format are .hdp, .jxr, and .wdp.

To work properly, D3DX11IFF\_WMP requires that you initialize COM. Therefore, call [CoInitialize](#) or [CoInitializeEx](#) in your application before you call D3DX.

## D3DX11IFF\_FORCE\_DWORD

Forces this enumeration to compile to 32 bits in size. Without this value, some compilers would allow this enumeration to compile to a size other than 32 bits. This value is not used.

## Remarks

See [Types of Bitmaps \(GDI+\)](#) for more information on some of these formats.

## Requirements

Requirement	Value
Header	D3DX11tex.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_NORMALMAP\_FLAG enumeration

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Normal map options. You can combine any number of these flags by using a bitwise OR operation.

## Syntax

C++

```
typedef enum D3DX11_NORMALMAP_FLAG {
    D3DX11_NORMALMAP_MIRROR_U          = (1 << 16),
    D3DX11_NORMALMAP_MIRROR_V          = (2 << 16),
    D3DX11_NORMALMAP_MIRROR           = (3 << 16),
    D3DX11_NORMALMAP_INVERTSIGN       = (8 << 16),
    D3DX11_NORMALMAP_COMPUTE_OCCLUSION = (16 << 16)
} D3DX11_NORMALMAP_FLAG, *LPD3DX11_NORMALMAP_FLAG;
```

## Constants

### D3DX11\_NORMALMAP\_MIRROR\_U

Indicates that pixels off the edge of the texture on the U-axis should be mirrored, not wrapped.

### D3DX11\_NORMALMAP\_MIRROR\_V

Indicates that pixels off the edge of the texture on the V-axis should be mirrored, not wrapped.

### D3DX11\_NORMALMAP\_MIRROR

Same as D3DX11\_NORMALMAP\_MIRROR\_U | D3DX11\_NORMALMAP\_MIRROR\_V.

### D3DX11\_NORMALMAP\_INVERTSIGN

Inverts the direction of each normal.

## D3DX11\_NORMALMAP\_COMPUTE\_OCCLUSION

Computes the per pixel occlusion term and encodes it into the alpha. An Alpha of 1 means that the pixel is not obscured in any way, and an alpha of 0 would mean that the pixel is completely obscured.

## Remarks

These flags are used by [D3DX11ComputeNormalMap](#).

## Requirements

Requirement	Value
Header	D3DX11tex.h

## See also

[D3DX Enumerations](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_SAVE\_TEXTURE\_FLAG enumeration

Article • 03/15/2021

## ⓘ Note

The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps.

Texture save options.

## Syntax

C++

```
typedef enum D3DX11_SAVE_TEXTURE_FLAG {
    D3DX11_STF_USEINPUTBLOB = 0x0001
} D3DX11_SAVE_TEXTURE_FLAG;
```

## Constants

**D3DX11\_STF\_USEINPUTBLOB**

Do not optimize.

## Requirements

Requirement	Value
Header	D3DX11tex.h

## See also

[D3DX Enumerations](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Effects 11 Reference

Article • 11/04/2020

Use Effects 11 source to build your effects-type application. The Effects 11 source is available at [Effects for Direct3D 11 Update](#). The Effects 11 API is described in this section.

## In this section

Topic	Description
<a href="#">Effects 11 Interfaces</a>	This section contains reference information for the component object model (COM) interfaces of the Effects 11 source.
<a href="#">Effects 11 Functions</a>	This section contains information about the Effects 11 functions.
<a href="#">Effects 11 Structures</a>	This section contains information about the Effects 11 structures.
<a href="#">Effect Format</a>	An effect (which is often stored in a file with a .fx file extension) declares the pipeline state set by an effect. Effect state can be roughly broken down into three categories:

## Related topics

[Direct3D 11 Reference](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Effects 11 Interfaces

Article • 11/04/2020

This section contains reference information for the component object model (COM) interfaces of the Effects 11 source.

## In this section

Topic	Description
<a href="#">ID3DX11Effect</a>	An <a href="#">ID3DX11Effect</a> interface manages a set of state objects, resources, and shaders for implementing a rendering effect.
<a href="#">ID3DX11EffectBlendVariable</a>	The blend-variable interface accesses blend state.
<a href="#">ID3DX11EffectClassInstanceVariable</a>	Accesses a class instance.
<a href="#">ID3DX11EffectConstantBuffer</a>	A constant-buffer interface accesses constant buffers or texture buffers.
<a href="#">ID3DX11EffectDepthStencilVariable</a>	A depth-stencil-variable interface accesses depth-stencil state.
<a href="#">ID3DX11EffectDepthStencilViewVariable</a>	A depth-stencil-view-variable interface accesses a depth-stencil view.
<a href="#">ID3DX11EffectGroup</a>	The <a href="#">ID3DX11EffectGroup</a> interface accesses an Effect group. The lifetime of an <a href="#">ID3DX11EffectGroup</a> object is equal to the lifetime of its parent <a href="#">ID3DX11Effect</a> object.
<a href="#">ID3DX11EffectInterfaceVariable</a>	Accesses an interface variable.
<a href="#">ID3DX11EffectMatrixVariable</a>	A matrix-variable interface accesses a matrix.
<a href="#">ID3DX11EffectPass</a>	An <a href="#">ID3DX11EffectPass</a> interface encapsulates state assignments within a technique. The lifetime of an <a href="#">ID3DX11EffectPass</a> object is equal to the lifetime of its parent <a href="#">ID3DX11Effect</a> object.
<a href="#">ID3DX11EffectRasterizerVariable</a>	A rasterizer-variable interface accesses rasterizer state.
<a href="#">ID3DX11EffectRenderTargetViewVariable</a>	A render-target-view interface accesses a render target.

Topic	Description
<a href="#">ID3DX11EffectSamplerVariable</a>	A sampler interface accesses sampler state.
<a href="#">ID3DX11EffectScalarVariable</a>	An effect-scalar-variable interface accesses scalar values.
<a href="#">ID3DX11EffectShaderResourceVariable</a>	A shader-resource interface accesses a shader resource.
<a href="#">ID3DX11EffectShaderVariable</a>	A shader-variable interface accesses a shader variable.
<a href="#">ID3DX11EffectStringVariable</a>	A string-variable interface accesses a string variable.
<a href="#">ID3DX11EffectTechnique</a>	<p>An <a href="#">ID3DX11EffectTechnique</a> interface is a collection of passes.</p> <p>The lifetime of an <a href="#">ID3DX11EffectTechnique</a> object is equal to the lifetime of its parent <a href="#">ID3DX11Effect</a> object.</p>
<a href="#">ID3DX11EffectType</a>	<p>The <a href="#">ID3DX11EffectType</a> interface accesses effect variables by type.</p> <p>The lifetime of an <a href="#">ID3DX11EffectType</a> object is equal to the lifetime of its parent <a href="#">ID3DX11Effect</a> object.</p>
<a href="#">ID3DX11EffectUnorderedAccessViewVariable</a>	Accesses an unordered access view.
<a href="#">ID3DX11EffectVariable</a>	<p>The <a href="#">ID3DX11EffectVariable</a> interface is the base class for all effect variables.</p> <p>The lifetime of an <a href="#">ID3DX11EffectVariable</a> object is equal to the lifetime of its parent <a href="#">ID3DX11Effect</a> object.</p>
<a href="#">ID3DX11EffectVectorVariable</a>	A vector-variable interface accesses a four-component vector.

## Related topics

[Effects 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ID3DX11Effect interface

Article • 04/26/2022

An **ID3DX11Effect** interface manages a set of state objects, resources, and shaders for implementing a rendering effect.

## Members

The **ID3DX11Effect** interface inherits from the [IUnknown](#) interface. **ID3DX11Effect** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11Effect** interface has these methods.

Method	Description
<a href="#">CloneEffect</a>	Creates a copy of an effect interface.
<a href="#">GetClassLinkage</a>	Gets a class linkage interface.
<a href="#">GetConstantBufferByIndex</a>	Get a constant buffer by index.
<a href="#">GetConstantBufferByName</a>	Get a constant buffer by name.
<a href="#">GetDesc</a>	Get an effect description.
<a href="#">GetDevice</a>	Get the device that created the effect.
<a href="#">GetGroupByIndex</a>	Gets an effect group by index.
<a href="#">GetGroupByName</a>	Gets an effect group by name.
<a href="#">GetTechniqueByIndex</a>	Get a technique by index.
<a href="#">GetTechniqueByName</a>	Get a technique by name.
<a href="#">GetVariableByIndex</a>	Get a variable by index.
<a href="#">GetVariableByName</a>	Get a variable by name.
<a href="#">GetVariableBySemantic</a>	Get a variable by semantic.
<a href="#">IsOptimized</a>	Test an effect to see if the reflection metadata has been removed from memory.

Method	Description
<a href="#">IsValid</a>	Test an effect to see if it contains valid syntax.
<a href="#">Optimize</a>	Minimize the amount of memory required for an effect.

## Remarks

An effect is created by calling [D3DX11CreateEffectFromMemory](#).

The effect system groups the information required for rendering into an effect which contains: state objects for assigning state changes in groups, resources for supplying input data and storing output data, and programs that control how the rendering is done called shaders.

### ① Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

### ① Note

If you call [QueryInterface](#) on an [ID3DX11Effect](#) object to retrieve the [IUnknown](#) interface, [QueryInterface](#) returns [E\\_NOINTERFACE](#). To work around this issue, use the following code:

```
IUnknown* pIUnknown = (IUnknown*)pEffect;          pIUnknown->AddRef();
```

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::CloneEffect method

Article • 03/15/2021

Creates a copy of an effect interface.

## Syntax

C++

```
HRESULT CloneEffect(  
    UINT          Flags,  
    ID3DX11Effect **ppClonedEffect  
) ;
```

## Parameters

*Flags*

Type: [UINT](#)

Flags affecting the creation of the cloned effect. Can be 0 or one of the following values.

Flag	Description
D3DX11_EFFECT_CLONE_FORCE_NONSINGLE	Ignore all "single" qualifiers on cbuffers. All cbuffers will have their own <a href="#">ID3D11Buffers</a> created in the cloned effect.

*ppClonedEffect*

Type: [ID3DX11Effect\\*\\*](#)

Pointer to an [ID3DX11Effect](#) pointer that will be set to the copy of the effect.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetClassLinkage method

Article • 03/15/2021

Gets a class linkage interface.

## Syntax

C++

```
ID3D11ClassLinkage* GetClassLinkage();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3D11ClassLinkage\\*](#)

Returns a pointer to an [ID3D11ClassLinkage](#) interface.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetConstantBufferByIndex method

Article • 03/15/2021

Get a constant buffer by index.

## Syntax

C++

```
ID3DX11EffectConstantBuffer* GetConstantBufferByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectConstantBuffer\\*](#)

A pointer to a [ID3DX11EffectConstantBuffer](#).

## Remarks

An effect that contains a variable that will be read/written by an application requires at least one constant buffer. For best performance, an effect should organize variables into one or more constant buffers based on their frequency of update.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetConstantBufferByName method

Article • 03/15/2021

Get a constant buffer by name.

## Syntax

C++

```
ID3DX11EffectConstantBuffer* GetConstantBufferByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

The constant-buffer name.

## Return value

Type: [ID3DX11EffectConstantBuffer\\*](#)

A pointer to the constant buffer indicated by the Name. See [ID3DX11EffectConstantBuffer](#).

## Remarks

An effect that contains a variable that will be read/written by an application requires at least one constant buffer. For best performance, an effect should organize variables into one or more constant buffers based on their frequency of update.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about

using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetDesc method

Article • 03/15/2021

Get an effect description.

## Syntax

C++

```
HRESULT GetDesc(  
    D3DX11_EFFECT_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_EFFECT\\_DESC\\*](#)

A pointer to an effect description (see [D3DX11\\_EFFECT\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An effect description contains basic information about an effect such as the techniques it contains and the constant buffer resources it requires.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetDevice method

Article • 03/15/2021

Get the device that created the effect.

## Syntax

C++

```
HRESULT GetDevice(  
    ID3D11Device **ppDevice  
) ;
```

## Parameters

*ppDevice*

Type: [ID3D11Device\\*\\*](#)

A pointer to an [ID3D11Device](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An effect is created for a specific device, by calling a function such as

[D3DX11CreateEffectFromMemory](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetGroupByIndex method

Article • 03/15/2021

Gets an effect group by index.

## Syntax

C++

```
ID3DX11EffectGroup* GetGroupByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index of the effect group.

## Return value

Type: [ID3DX11EffectGroup\\*](#)

A pointer to an [ID3DX11EffectGroup](#) interface.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetGroupByName method

Article • 03/15/2021

Gets an effect group by name.

## Syntax

C++

```
ID3DX11EffectGroup* GetGroupByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

Name of the effect group.

## Return value

Type: [ID3DX11EffectGroup\\*](#)

A pointer to an [ID3DX11EffectGroup](#) interface.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetTechniqueByIndex method

Article • 03/15/2021

Get a technique by index.

## Syntax

C++

```
ID3DX11EffectTechnique* GetTechniqueByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectTechnique\\*](#)

A pointer to an [ID3DX11EffectTechnique](#).

## Remarks

An effect contains one or more techniques; each technique contains one or more passes. You can access a technique using its name or with an index.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetTechniqueByName method

Article • 03/15/2021

Get a technique by name.

## Syntax

C++

```
ID3DX11EffectTechnique* GetTechniqueByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

The name of the technique.

## Return value

Type: [ID3DX11EffectTechnique\\*](#)

A pointer to an [ID3DX11EffectTechnique](#). If a technique with the appropriate name is not found an invalid technique is returned. [ID3DX11EffectTechnique::IsValid](#) should be called on the returned technique to determine whether it is valid.

## Remarks

An effect contains one or more techniques; each technique contains one or more passes. You can access a technique using its name or with an index.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about

using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetVariableByIndex method

Article • 03/15/2021

Get a variable by index.

## Syntax

C++

```
ID3DX11EffectVariable* GetVariableByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to a [ID3DX11EffectVariable](#).

## Remarks

An effect may contain one or more variables. Variables outside of a technique are considered global to all effects, those located inside of a technique are local to that technique. You can access any local non-static effect variable using its name or with an index.

The method returns a pointer to an [effect-variable interface](#) if a variable is not found; you can call [ID3DX11Effect::IsValid](#) to verify whether or not the index exists.

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetVariableByName method

Article • 03/15/2021

Get a variable by name.

## Syntax

C++

```
ID3DX11EffectVariable* GetVariableByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

The variable name.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#). Returns an invalid variable if the specified name cannot be found.

## Remarks

An effect may contain one or more variables. Variables outside of a technique are considered global to all effects, those located inside of a technique are local to that technique. You can access an effect variable using its name or with an index.

The method returns a pointer to an [effect-variable interface](#) whether or not a variable is found. [ID3DX11Effect::IsValid](#) should be called to verify whether or not the name exists.

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::GetVariableBySemantic method

Article • 03/15/2021

Get a variable by semantic.

## Syntax

C++

```
ID3DX11EffectVariable* GetVariableBySemantic(  
    LPCSTR Semantic  
);
```

## Parameters

*Semantic*

Type: [LPCSTR](#)

The semantic name.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to the effect variable indicated by the Semantic. See [ID3DX11EffectVariable](#).

## Remarks

Each effect variable can have a semantic attached, which is a user defined metadata string. Some [system-value semantics](#) are reserved words that trigger built in functionality by pipeline stages.

The method returns a pointer to an [effect-variable interface](#) if a variable is not found; you can call [ID3DX11Effect::IsValid](#) to verify whether or not the semantic exists.

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::IsOptimized method

Article • 03/15/2021

Test an effect to see if the reflection metadata has been removed from memory.

## Syntax

C++

```
BOOL IsOptimized();
```

## Parameters

This method has no parameters.

## Return value

Type: **BOOL**

TRUE if the effect is optimized; otherwise FALSE.

## Remarks

An effect uses memory space two different ways: to store the information required by the runtime to execute an effect, and to store the metadata required to reflect information back to an application using the API. You can minimize the amount of memory required by an effect by calling [ID3DX11Effect::Optimize](#) which removes the reflection metadata from memory. Of course, API methods to read variables will no longer work once reflection data has been removed.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::IsValid method

Article • 03/15/2021

Test an effect to see if it contains valid syntax.

## Syntax

C++

```
BOOL IsValid();
```

## Parameters

This method has no parameters.

## Return value

Type: **BOOL**

TRUE if the code syntax is valid; otherwise FALSE.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11Effect::Optimize method

Article • 03/15/2021

Minimize the amount of memory required for an effect.

## Syntax

C++

```
HRESULT Optimize();
```

## Parameters

This method has no parameters.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An effect uses memory space two different ways: to store the information required by the runtime to execute an effect, and to store the metadata required to reflect information back to an application using the API. You can minimize the amount of memory required by an effect by calling **ID3DX11Effect::Optimize** which removes the reflection metadata from memory. API methods to read variables will no longer work once reflection data has been removed.

The following methods will fail after Optimize has been called on an effect.

- [ID3DX11Effect::GetConstantBufferByIndex](#)
- [ID3DX11Effect::GetConstantBufferByName](#)
- [ID3DX11Effect::GetDesc](#)
- [ID3DX11Effect::GetDevice](#)
- [ID3DX11Effect::GetTechniqueByIndex](#)
- [ID3DX11Effect::GetTechniqueByName](#)
- [ID3DX11Effect::GetVariableByIndex](#)

- [ID3DX11Effect::GetVariableByName](#)
- [ID3DX11Effect::GetVariableBySemantic](#)

ⓘ Note

References retrieved with these methods before calling [ID3DX11Effect::Optimize](#) are still valid after [ID3DX11Effect::Optimize](#) is called. This allows the application to get all the variables, techniques, and passes it will use, call Optimize, and then use the effect.

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11Effect](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectBlendVariable interface

Article • 03/15/2021

The blend-variable interface accesses blend state.

## Members

The **ID3DX11EffectBlendVariable** interface inherits from [ID3DX11EffectVariable](#).

**ID3DX11EffectBlendVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectBlendVariable** interface has these methods.

Method	Description
<a href="#">GetBackingStore</a>	Get a pointer to a blend-state variable.
<a href="#">GetBlendState</a>	Get a pointer to a blend-state interface.
<a href="#">SetBlendState</a>	Sets an effect's blend-state.
<a href="#">UndoSetBlendState</a>	Reverts a previously set blend-state.

## Remarks

An [ID3DX11EffectVariable](#) interface is created when an effect is read into memory.

Effect variables are saved in memory in the backing store; when a technique is applied, the values in the backing store are copied to the device. You can use either of these methods to return state.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectBlendVariable::GetBackingStore method

Article • 03/15/2021

Get a pointer to a blend-state variable.

## Syntax

C++

```
HRESULT GetBackingStore(
    UINT           Index,
    D3D11_BLEND_DESC *pBlendDesc
);
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of blend-state descriptions. If there is only one blend-state variable in the effect, use 0.

*pBlendDesc*

Type: [D3D11\\_BLEND\\_DESC\\*](#)

A pointer to a blend-state description (see [D3D11\\_BLEND\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Effect variables are saved in memory in the backing store; when a technique is applied, the values in the backing store are copied to the device. Backing store data can be used to

recreate the variable when necessary.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectBlendVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectBlendVariable::GetBlendState method

Article • 03/15/2021

Get a pointer to a blend-state interface.

## Syntax

C++

```
HRESULT GetBlendState(  
    UINT             Index,  
    ID3D11BlendState **ppBlendState  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of blend-state interfaces. If there is only one blend-state interface, use 0.

*ppBlendState*

Type: [ID3D11BlendState\\*\\*](#)

The address of a pointer to a blend-state interface (see [ID3D11BlendState](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectBlendVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectBlendVariable::SetBlendState method

Article • 03/15/2021

Sets an effect's blend-state.

## Syntax

C++

```
HRESULT SetBlendState(  
    UINT             Index,  
    ID3D11BlendState *pBlendState  
>;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of blend-state interfaces. If there is only one blend-state interface, use 0.

*pBlendState*

Type: [ID3D11BlendState\\*](#)

A pointer to an [ID3D11BlendState](#) interface containing the blend-state to set.

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectBlendVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectBlendVariable::UndoSetBlendState method

Article • 03/15/2021

Reverts a previously set blend-state.

## Syntax

C++

```
HRESULT UndoSetBlendState(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of blend-state interfaces. If there is only one blend-state interface, use 0.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectBlendVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectClassInstanceVariable interface

Article • 03/15/2021

Accesses a class instance.

## Members

The **ID3DX11EffectClassInstanceVariable** interface inherits from [ID3DX11EffectVariable](#). **ID3DX11EffectClassInstanceVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectClassInstanceVariable** interface has these methods.

Method	Description
<a href="#">GetClassInstance</a>	Gets a class instance.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectClassInstanceVariable::GetClassInstance method

Article • 03/15/2021

Gets a class instance.

## Syntax

C++

```
HRESULT GetClassInstance(  
    ID3D11ClassInstance **ppClassInstance  
) ;
```

## Parameters

*ppClassInstance*

Type: [ID3D11ClassInstance\\*\\*](#)

Pointer to an [ID3D11ClassInstance](#) pointer that will be set to class instance.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectClassInstanceVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectConstantBuffer interface

Article • 03/15/2021

A constant-buffer interface accesses constant buffers or texture buffers.

## Members

The **ID3DX11EffectConstantBuffer** interface inherits from [ID3DX11EffectVariable](#).

**ID3DX11EffectConstantBuffer** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectConstantBuffer** interface has these methods.

Method	Description
<a href="#">GetConstantBuffer</a>	Get a constant-buffer.
<a href="#">GetTextureBuffer</a>	Get a texture-buffer.
<a href="#">SetConstantBuffer</a>	Set a constant-buffer.
<a href="#">SetTextureBuffer</a>	Set a texture-buffer.
<a href="#">UndoSetConstantBuffer</a>	Reverts a previously set constant buffer.
<a href="#">UndoSetTextureBuffer</a>	Reverts a previously set texture buffer.

## Remarks

Use constant buffers to store many effect constants; grouping constants into buffers based on their frequency of update. This allows you to minimize the number of state changes as well as make the fewest API calls to change state. Both of these factors lead to better performance.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectConstantBuffer::GetConstantBuffer method

Article • 03/15/2021

Get a constant-buffer.

## Syntax

C++

```
HRESULT GetConstantBuffer(  
    ID3D11Buffer **ppConstantBuffer  
) ;
```

## Parameters

*ppConstantBuffer*

Type: [ID3D11Buffer\\*\\*](#)

The address of a pointer to a constant-buffer interface. See [ID3D11Buffer](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectConstantBuffer](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectConstantBuffer::GetTextureBuffer method

Article • 03/15/2021

Get a texture-buffer.

## Syntax

C++

```
HRESULT GetTextureBuffer(  
    ID3D11ShaderResourceView **ppTextureBuffer  
);
```

## Parameters

*ppTextureBuffer*

Type: [ID3D11ShaderResourceView\\*\\*](#)

The address of a pointer to a shader-resource-view interface for accessing a texture buffer. See [ID3D11ShaderResourceView](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectConstantBuffer](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectConstantBuffer::SetConstantBuffer method

Article • 03/15/2021

Set a constant-buffer.

## Syntax

C++

```
HRESULT SetConstantBuffer(  
    ID3D11Buffer *pConstantBuffer  
>;
```

## Parameters

*pConstantBuffer*

Type: [ID3D11Buffer\\*](#)

A pointer to a constant-buffer interface. See [ID3D11Buffer](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectConstantBuffer](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectConstantBuffer::SetTextureBuffer method

Article • 03/15/2021

Set a texture-buffer.

## Syntax

C++

```
HRESULT SetTextureBuffer(  
    ID3D11ShaderResourceView *pTextureBuffer  
>;
```

## Parameters

*pTextureBuffer*

Type: [ID3D11ShaderResourceView\\*](#)

A pointer to a shader-resource-view interface for accessing a texture buffer.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectConstantBuffer](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectConstantBuffer::UndoSetConstantBuffer method

Article • 03/15/2021

Reverts a previously set constant buffer.

## Syntax

C++

```
HRESULT UndoSetConstantBuffer();
```

## Parameters

This method has no parameters.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectConstantBuffer](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DX11EffectConstantBuffer::UndoSetTextureBuffer method

Article • 03/15/2021

Reverts a previously set texture buffer.

## Syntax

C++

```
HRESULT UndoSetTextureBuffer();
```

## Parameters

This method has no parameters.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectConstantBuffer](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilVariable interface

Article • 03/15/2021

A depth-stencil-variable interface accesses depth-stencil state.

## Members

The **ID3DX11EffectDepthStencilVariable** interface inherits from [ID3DX11EffectVariable](#). **ID3DX11EffectDepthStencilVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectDepthStencilVariable** interface has these methods.

Method	Description
<a href="#">GetBackingStore</a>	Get a pointer to a variable that contains depth-stencil state.
<a href="#">GetDepthStencilState</a>	Get a pointer to a depth-stencil interface.
<a href="#">SetDepthStencilState</a>	Sets the depth stencil state.
<a href="#">UndoSetDepthStencilState</a>	Reverts a previously set depth stencil state.

## Remarks

An [ID3DX11EffectVariable](#) interface is created when an effect is read into memory.

Effect variables are saved in memory in the backing store; when a technique is applied, the values in the backing store are copied to the device. You can use either of these methods to return state.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilVariable::GetBackingStore method

Article • 03/15/2021

Get a pointer to a variable that contains depth-stencil state.

## Syntax

C++

```
HRESULT GetBackingStore(  
    UINT             Index,  
    D3D11_DEPTH_STENCIL_DESC *pDepthStencilDesc  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of depth-stencil-state descriptions. If there is only one depth-stencil variable in the effect, use 0.

*pDepthStencilDesc*

Type: [D3D11\\_DEPTH\\_STENCIL\\_DESC\\*](#)

A pointer to a depth-stencil-state description (see [D3D11\\_DEPTH\\_STENCIL\\_DESC](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Effect variables are saved in memory in the backing store; when a technique is applied, the values in the backing store are copied to the device. Backing store data can be used to

recreate the variable when necessary.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectDepthStencilVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilVariable::GetDepthStencilState method

Article • 03/15/2021

Get a pointer to a depth-stencil interface.

## Syntax

C++

```
HRESULT GetDepthStencilState(  
    UINT             Index,  
    ID3D11DepthStencilState **ppDepthStencilState  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of depth-stencil interfaces. If there is only one depth-stencil interface, use 0.

*ppDepthStencilState*

Type: [ID3D11DepthStencilState\\*\\*](#)

The address of a pointer to a blend-state interface (see [ID3D11DepthStencilState](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectDepthStencilVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilVariable::SetDepthStencilState method

Article • 03/15/2021

Sets the depth stencil state.

## Syntax

C++

```
HRESULT SetDepthStencilState(  
    UINT             Index,  
    ID3D11DepthStencilState *pDepthStencilState  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of depth-stencil interfaces. If there is only one depth-stencil interface, use 0.

*pDepthStencilState*

Type: [ID3D11DepthStencilState\\*](#)

Pointer to an [ID3D11DepthStencilState](#) interface containing the new depth stencil state.

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectDepthStencilVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilVariable::UndoSetDepthStencilState method

Article • 03/15/2021

Reverts a previously set depth stencil state.

## Syntax

C++

```
HRESULT UndoSetDepthStencilState(  
    UINT Index  
);
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of depth-stencil interfaces. If there is only one depth-stencil interface, use 0.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectDepthStencilVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilViewVariable interface

Article • 03/15/2021

A depth-stencil-view-variable interface accesses a depth-stencil view.

## Members

The **ID3DX11EffectDepthStencilViewVariable** interface inherits from [ID3DX11EffectVariable](#). **ID3DX11EffectDepthStencilViewVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectDepthStencilViewVariable** interface has these methods.

Method	Description
<a href="#">GetDepthStencil</a>	Get a depth-stencil-view resource.
<a href="#">GetDepthStencilArray</a>	Get an array of depth-stencil-view resources.
<a href="#">SetDepthStencil</a>	Set a depth-stencil-view resource.
<a href="#">SetDepthStencilArray</a>	Set an array of depth-stencil-view resources.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilViewVariable::GetDepthStencil method

Article • 03/15/2021

Get a depth-stencil-view resource.

## Syntax

C++

```
HRESULT GetDepthStencil(  
    ID3D11DepthStencilView **ppResource  
) ;
```

## Parameters

*ppResource*

Type: [ID3D11DepthStencilView\\*\\*](#)

The address of a pointer to a depth-stencil-view interface. See [ID3D11DepthStencilView](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectDepthStencilViewVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilViewVariable::GetDepthStencilArray method

Article • 03/15/2021

Get an array of depth-stencil-view resources.

## Syntax

C++

```
HRESULT GetDepthStencilArray(
    ID3D11DepthStencilView **ppResources,
    UINT                 Offset,
    UINT                 Count
);
```

## Parameters

*ppResources*

Type: [ID3D11DepthStencilView\\*\\*](#)

A pointer to an array of depth-stencil-view interfaces. See [ID3D11DepthStencilView](#).

*Offset*

Type: [UINT](#)

The zero-based array index to get the first interface.

*Count*

Type: [UINT](#)

The number of elements in the array.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectDepthStencilViewVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilViewVariable::SetDepthStencil method

Article • 03/15/2021

Set a depth-stencil-view resource.

## Syntax

C++

```
HRESULT SetDepthStencil(  
    ID3D11DepthStencilView *pResource  
) ;
```

## Parameters

*pResource*

Type: [ID3D11DepthStencilView\\*](#)

A pointer to a depth-stencil-view interface. See [ID3D11DepthStencilView](#).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectDepthStencilViewVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectDepthStencilViewVariable::SetDepthStencilArray method

Article • 03/15/2021

Set an array of depth-stencil-view resources.

## Syntax

C++

```
HRESULT SetDepthStencilArray(
    ID3D11DepthStencilView **ppResources,
    UINT                 Offset,
    UINT                 Count
);
```

## Parameters

*ppResources*

Type: [ID3D11DepthStencilView\\*\\*](#)

A pointer to an array of depth-stencil-view interfaces. See [ID3D11DepthStencilView](#).

*Offset*

Type: [UINT](#)

The zero-based array index to set the first interface.

*Count*

Type: [UINT](#)

The number of elements in the array.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectDepthStencilViewVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectGroup interface

Article • 03/15/2021

The **ID3DX11EffectGroup** interface accesses an Effect group.

The lifetime of an **ID3DX11EffectGroup** object is equal to the lifetime of its parent **ID3DX11Effect** object.

- [Methods](#)

## Methods

The **ID3DX11EffectGroup** interface has these methods.

Method	Description
<a href="#">GetAnnotationByIndex</a>	Get an annotation by index.
<a href="#">GetAnnotationByName</a>	Get an annotation by name.
<a href="#">GetDesc</a>	Gets a group description.
<a href="#">GetTechniqueByIndex</a>	Get a technique by index.
<a href="#">GetTechniqueByName</a>	Get a technique by name.
<a href="#">IsValid</a>	Test an effect to see if it contains valid syntax.

## Remarks

To get an **ID3DX11EffectGroup** interface, call a method like

[ID3DX11Effect::GetGroupByName](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DX11EffectGroup::GetAnnotationByIndex method

Article • 03/15/2021

Get an annotation by index.

## Syntax

C++

```
ID3DX11EffectVariable* GetAnnotationByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index of the annotation.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

Pointer to an [ID3DX11EffectVariable](#) interface.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectGroup](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectGroup::GetAnnotationByName method

Article • 03/15/2021

Get an annotation by name.

## Syntax

C++

```
ID3DX11EffectVariable* GetAnnotationByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

The name of the annotation.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#). Note that if the annotation is not found the [ID3DX11EffectVariable](#) returned will be empty. The [ID3DX11EffectVariable::IsValid](#) method should be called to determine whether the annotation was found.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectGroup](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectGroup::GetDesc method

Article • 03/15/2021

Gets a group description.

## Syntax

C++

```
HRESULT GetDesc(  
    D3DX11_GROUP_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_GROUP\\_DESC\\*](#)

A pointer to a [D3DX11\\_GROUP\\_DESC](#) structure.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
-------------	-------

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectGroup](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectGroup::GetTechniqueByIndex method

Article • 03/15/2021

Get a technique by index.

## Syntax

C++

```
ID3DX11EffectTechnique* GetTechniqueByIndex(  
    UINT Index  
);
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectTechnique\\*](#)

A pointer to an [ID3DX11EffectTechnique](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectGroup](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectGroup::GetTechniqueByName method

Article • 03/15/2021

Get a technique by name.

## Syntax

C++

```
ID3DX11EffectTechnique* GetTechniqueByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

The name of the technique.

## Return value

Type: [ID3DX11EffectTechnique\\*](#)

A pointer to an [ID3DX11EffectTechnique](#), or **NULL** if the technique is not found.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectGroup](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectGroup::IsValid method

Article • 03/15/2021

Test an effect to see if it contains valid syntax.

## Syntax

C++

```
BOOL IsValid();
```

## Parameters

This method has no parameters.

## Return value

Type: **BOOL**

TRUE if the code syntax is valid; otherwise FALSE.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectGroup](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectInterfaceVariable interface

Article • 03/15/2021

Accesses an interface variable.

## Members

The **ID3DX11EffectInterfaceVariable** interface inherits from [ID3DX11EffectVariable](#). **ID3DX11EffectInterfaceVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectInterfaceVariable** interface has these methods.

Method	Description
<a href="#">GetClassInstance</a>	Get a class instance.
<a href="#">SetClassInstance</a>	Sets a class instance.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectInterfaceVariable::GetClassInstance method

Article • 03/15/2021

Get a class instance.

## Syntax

C++

```
HRESULT GetClassInstance(
    ID3DX11EffectClassInstanceVariable **ppEffectClassInstance
);
```

## Parameters

*ppEffectClassInstance*

Type: [ID3DX11EffectClassInstanceVariable\\*\\*](#)

Pointer to an [ID3DX11EffectClassInstanceVariable](#) pointer that will be set to the class instance.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectInterfaceVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectInterfaceVariable::SetClassInstance method

Article • 03/15/2021

Sets a class instance.

## Syntax

C++

```
HRESULT SetClassInstance(  
    ID3DX11EffectClassInstanceVariable *pEffectClassInstance  
>;
```

## Parameters

*pEffectClassInstance*

Type: [ID3DX11EffectClassInstanceVariable\\*](#)

Pointer to an [ID3DX11EffectClassInstanceVariable](#) interface.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectInterfaceVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable interface

Article • 03/15/2021

A matrix-variable interface accesses a matrix.

## Members

The **ID3DX11EffectMatrixVariable** interface inherits from [ID3DX11EffectVariable](#).  
**ID3DX11EffectMatrixVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectMatrixVariable** interface has these methods.

Method	Description
<a href="#">GetMatrix</a>	Get a matrix.
<a href="#">GetMatrixArray</a>	Get an array of matrices.
<a href="#">GetMatrixTranspose</a>	Transpose and get a floating-point matrix.
<a href="#">GetMatrixTransposeArray</a>	Transpose and get an array of floating-point matrices.
<a href="#">SetMatrix</a>	Set a floating-point matrix.
<a href="#">SetMatrixArray</a>	Set an array of floating-point matrices.
<a href="#">SetMatrixTranspose</a>	Transpose and set a floating-point matrix.
<a href="#">SetMatrixTransposeArray</a>	Transpose and set an array of floating-point matrices.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable::GetMatrix method

Article • 03/15/2021

Get a matrix.

## Syntax

C++

```
HRESULT GetMatrix(  
    float *pData  
>;
```

## Parameters

*pData*

Type: **float\***

A pointer to the first element in a matrix.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectMatrixVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable::GetMatrixArray method

Article • 03/15/2021

Get an array of matrices.

## Syntax

C++

```
HRESULT GetMatrixArray(  
    float *pData,  
    UINT  Offset,  
    UINT  Count  
>;
```

## Parameters

*pData*

Type: **float\***

A pointer to the first element of the first matrix in an array of matrices.

*Offset*

Type: **UINT**

The offset (in number of matrices) between the start of the array and the first matrix returned.

*Count*

Type: **UINT**

The number of matrices in the returned array.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectMatrixVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable::GetMatrixTranspose method

Article • 03/15/2021

Transpose and get a floating-point matrix.

## Syntax

C++

```
HRESULT GetMatrixTranspose(  
    float *pData  
);
```

## Parameters

*pData*

Type: **float\***

A pointer to the first element of a transposed matrix.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Transposing a matrix will rearrange the data order from row-column order to column-row order (or vice versa).

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectMatrixVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable::GetMatrixTransposeArray method

Article • 03/15/2021

Transpose and get an array of floating-point matrices.

## Syntax

C++

```
HRESULT GetMatrixTransposeArray(  
    float *pData,  
    UINT  Offset,  
    UINT  Count  
>;
```

## Parameters

*pData*

Type: **float\***

A pointer to the first element of an array of tranposed matrices.

*Offset*

Type: **UINT**

The offset (in number of matrices) between the start of the array and the first matrix to get.

*Count*

Type: **UINT**

The number of matrices in the array to get.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Transposing a matrix will rearrange the data order from row-column order to column-row order (or vice versa).

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectMatrixVariable](#)

## Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable::SetMatrix method

Article • 03/15/2021

Set a floating-point matrix.

## Syntax

C++

```
HRESULT SetMatrix(  
    float *pData  
) ;
```

## Parameters

*pData*

Type: **float\***

A pointer to the first element in the matrix.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectMatrixVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable::SetMatrixArray method

Article • 03/15/2021

Set an array of floating-point matrices.

## Syntax

C++

```
HRESULT SetMatrixArray(  
    float *pData,  
    UINT Offset,  
    UINT Count  
)
```

## Parameters

*pData*

Type: **float\***

A pointer to the first matrix.

*Offset*

Type: **UINT**

The number of matrix elements to skip from the start of the array.

*Count*

Type: **UINT**

The number of elements to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectMatrixVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable::SetMatrixTranspose method

Article • 03/15/2021

Transpose and set a floating-point matrix.

## Syntax

C++

```
HRESULT SetMatrixTranspose(  
    float *pData  
>;
```

## Parameters

*pData*

Type: **float\***

A pointer to the first element of a matrix.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Transposing a matrix will rearrange the data order from row-column order to column-row order (or vice versa).

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectMatrixVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectMatrixVariable::SetMatrixTransposeArray method

Article • 03/15/2021

Transpose and set an array of floating-point matrices.

## Syntax

C++

```
HRESULT SetMatrixTransposeArray(  
    float *pData,  
    UINT Offset,  
    UINT Count  
)
```

## Parameters

*pData*

Type: **float\***

A pointer to an array of matrices.

*Offset*

Type: **UINT**

The offset (in number of matrices) between the start of the array and the first matrix to set.

*Count*

Type: **UINT**

The number of matrices in the array to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Transposing a matrix will rearrange the data order from row-column order to column-row order (or vice versa).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectMatrixVariable](#)

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass interface

Article • 03/15/2021

An **ID3DX11EffectPass** interface encapsulates state assignments within a technique.

The lifetime of an **ID3DX11EffectPass** object is equal to the lifetime of its parent **ID3DX11Effect** object.

- [Methods](#)

## Methods

The **ID3DX11EffectPass** interface has these methods.

Method	Description
<a href="#">Apply</a>	Set the state contained in a pass to the device.
<a href="#">ComputeStateBlockMask</a>	Generate a mask for allowing/preventing state changes.
<a href="#">GetAnnotationByIndex</a>	Get an annotation by index.
<a href="#">GetAnnotationByName</a>	Get an annotation by name.
<a href="#">GetComputeShaderDesc</a>	Get a compute-shader description.
<a href="#">GetDesc</a>	Get a pass description.
<a href="#">GetDomainShaderDesc</a>	Get a domain-shader description.
<a href="#">GetGeometryShaderDesc</a>	Get a geometry-shader description.
<a href="#">GetHullShaderDesc</a>	Get hull-shader description.
<a href="#">GetPixelShaderDesc</a>	Get a pixel-shader description.
<a href="#">GetVertexShaderDesc</a>	Get a vertex-shader description.
<a href="#">IsValid</a>	Test a pass to see if it contains valid syntax.

## Remarks

A pass is a block of code that sets render-state objects and shaders. A pass is declared within a technique.

To get an effect-pass interface, call a method like [ID3DX11EffectTechnique::GetPassByName](#).

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::Apply method

Article • 03/15/2021

Set the state contained in a pass to the device.

## Syntax

C++

```
HRESULT Apply(  
    UINT           Flags,  
    ID3D11DeviceContext *pContext  
>;
```

## Parameters

*Flags*

Type: [UINT](#)

Unused.

*pContext*

Type: [ID3D11DeviceContext\\*](#)

The [ID3D11DeviceContext](#) to apply the pass to.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::ComputeStateBlockMask method

Article • 03/15/2021

Generate a mask for allowing/preventing state changes.

## Syntax

C++

```
HRESULT ComputeStateBlockMask(  
    D3DX11_STATE_BLOCK_MASK *pStateBlockMask  
) ;
```

## Parameters

*pStateBlockMask*

Type: [D3DX11\\_STATE\\_BLOCK\\_MASK\\*](#)

A pointer to a state-block mask (see [D3DX11\\_STATE\\_BLOCK\\_MASK](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetAnnotationByIndex method

Article • 03/15/2021

Get an annotation by index.

## Syntax

C++

```
ID3DX11EffectVariable* GetAnnotationByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetAnnotationByName method

Article • 03/15/2021

Get an annotation by name.

## Syntax

C++

```
ID3DX11EffectVariable* GetAnnotationByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

The name of the annotation.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetComputeShaderDesc method

Article • 03/15/2021

Get a compute-shader description.

## Syntax

C++

```
HRESULT GetComputeShaderDesc(  
    D3DX11_PASS_SHADER_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_PASS\\_SHADER\\_DESC\\*](#)

A pointer to a compute-shader description (see [D3DX11\\_PASS\\_SHADER\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetDesc method

Article • 03/15/2021

Get a pass description.

## Syntax

C++

```
HRESULT GetDesc(  
    D3DX11_PASS_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_PASS\\_DESC\\*](#)

A pointer to a pass description (see [D3DX11\\_PASS\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

A pass is a block of code that sets render state and shaders (which in turn sets constant buffers, samplers and textures). An effect technique contains one or more passes.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetDomainShaderDesc method

Article • 03/15/2021

Get a domain-shader description.

## Syntax

C++

```
HRESULT GetDomainShaderDesc(  
    D3DX11_PASS_SHADER_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_PASS\\_SHADER\\_DESC\\*](#)

A pointer to a domain-shader description (see [D3DX11\\_PASS\\_SHADER\\_DESC](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetGeometryShaderDesc method

Article • 03/15/2021

Get a geometry-shader description.

## Syntax

C++

```
HRESULT GetGeometryShaderDesc(  
    D3DX11_PASS_SHADER_DESC *pDesc  
);
```

## Parameters

*pDesc*

Type: [D3DX11\\_PASS\\_SHADER\\_DESC\\*](#)

A pointer to a geometry-shader description (see [D3DX11\\_PASS\\_SHADER\\_DESC](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An effect pass can contain render state assignments and shader object assignments.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetHullShaderDesc method

Article • 03/15/2021

Get hull-shader description.

## Syntax

C++

```
HRESULT GetHullShaderDesc(  
    D3DX11_PASS_SHADER_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_PASS\\_SHADER\\_DESC\\*](#)

A pointer to a hull-shader description (see [D3DX11\\_PASS\\_SHADER\\_DESC](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetPixelShaderDesc method

Article • 03/15/2021

Get a pixel-shader description.

## Syntax

C++

```
HRESULT GetPixelShaderDesc(  
    D3DX11_PASS_SHADER_DESC *pDesc  
);
```

## Parameters

*pDesc*

Type: [D3DX11\\_PASS\\_SHADER\\_DESC\\*](#)

A pointer to a pixel-shader description (see [D3DX11\\_PASS\\_SHADER\\_DESC](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An effect pass can contain render state assignments and shader object assignments.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::GetVertexShaderDesc method

Article • 03/15/2021

Get a vertex-shader description.

## Syntax

C++

```
HRESULT GetVertexShaderDesc(  
    D3DX11_PASS_SHADER_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_PASS\\_SHADER\\_DESC\\*](#)

A pointer to a vertex-shader description (see [D3DX11\\_PASS\\_SHADER\\_DESC](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

An effect pass can contain render state assignments and shader object assignments.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectPass::IsValid method

Article • 03/15/2021

Test a pass to see if it contains valid syntax.

## Syntax

C++

```
BOOL IsValid();
```

## Parameters

This method has no parameters.

## Return value

Type: **BOOL**

TRUE if the code syntax is valid; otherwise FALSE.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectPass](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRasterizerVariable interface

Article • 03/15/2021

A rasterizer-variable interface accesses rasterizer state.

## Members

The **ID3DX11EffectRasterizerVariable** interface inherits from [ID3DX11EffectVariable](#). **ID3DX11EffectRasterizerVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectRasterizerVariable** interface has these methods.

Method	Description
<a href="#">GetBackingStore</a>	Get a pointer to a variable that contains rasteriser state.
<a href="#">GetRasterizerState</a>	Get a pointer to a rasterizer interface.
<a href="#">SetRasterizerState</a>	Sets the rasterizer state.
<a href="#">UndoSetRasterizerState</a>	Reverts a previously set rasterizer state.

## Remarks

An [ID3DX11EffectVariable](#) interface is created when an effect is read into memory.

Effect variables are saved in memory in the backing store; when a technique is applied, the values in the backing store are copied to the device. You can use either of these methods to return state.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRasterizerVariable::GetBackingStore method

Article • 03/15/2021

Get a pointer to a variable that contains rasteriser state.

## Syntax

C++

```
HRESULT GetBackingStore(  
    UINT             Index,  
    D3D11_RASTERIZER_DESC *pRasterizerDesc  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of rasteriser-state descriptions. If there is only one rasteriser variable in the effect, use 0.

*pRasterizerDesc*

Type: [D3D11\\_RASTERIZER\\_DESC\\*](#)

A pointer to a rasteriser-state description (see [D3D11\\_RASTERIZER\\_DESC](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

Effect variables are saved in memory in the backing store; when a technique is applied, the values in the backing store are copied to the device. Backing store data can used to

recreate the variable when necessary.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRasterizerVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRasterizerVariable::GetRasterizerState method

Article • 03/15/2021

Get a pointer to a rasterizer interface.

## Syntax

C++

```
HRESULT GetRasterizerState(  
    UINT           Index,  
    ID3D11RasterizerState **ppRasterizerState  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of rasterizer interfaces. If there is only one rasterizer interface, use 0.

*ppRasterizerState*

Type: [ID3D11RasterizerState\\*\\*](#)

The address of a pointer to a rasterizer interface (see [ID3D11RasterizerState](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRasterizerVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRasterizerVariable::SetRasterizerState method

Article • 03/15/2021

Sets the rasterizer state.

## Syntax

C++

```
HRESULT SetRasterizerState(  
    UINT           Index,  
    ID3D11RasterizerState *pRasterizerState  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of rasterizer interfaces. If there is only one rasterizer interface, use 0.

*pRasterizerState*

Type: [ID3D11RasterizerState\\*](#)

Pointer to an [ID3D11RasterizerState](#) interface.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRasterizerVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRasterizerVariable::UndoSetRasterizerState method

Article • 03/15/2021

Reverts a previously set rasterizer state.

## Syntax

C++

```
HRESULT UndoSetRasterizerState(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of rasterizer interfaces. If there is only one rasterizer interface, use 0.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRasterizerVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRenderTargetViewVariable interface

Article • 03/15/2021

A render-target-view interface accesses a render target.

## Members

The **ID3DX11EffectRenderTargetViewVariable** interface inherits from [ID3DX11EffectRasterizerVariable](#). **ID3DX11EffectRenderTargetViewVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectRenderTargetViewVariable** interface has these methods.

Method	Description
<a href="#">GetRenderTarget</a>	Get a render-target.
<a href="#">GetRenderTargetArray</a>	Get an array of render-targets.
<a href="#">SetRenderTarget</a>	Set a render-target.
<a href="#">SetRenderTargetArray</a>	Set an array of render-targets.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRasterizerVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRenderTargetViewVariable::GetRenderTarget method

Article • 03/15/2021

Get a render-target.

## Syntax

C++

```
HRESULT GetRenderTarget(  
    ID3D11RenderTargetView **ppResource  
>;
```

## Parameters

*ppResource*

Type: [ID3D11RenderTargetView\\*\\*](#)

The address of a pointer to a render-target-view interface. See [ID3D11RenderTargetView](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRenderTargetViewVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRenderTargetViewVariable::GetRenderTargetArray method

Article • 03/15/2021

Get an array of render-targets.

## Syntax

C++

```
HRESULT GetRenderTargetArray(
    ID3D11RenderTargetView **ppResources,
    UINT                 Offset,
    UINT                 Count
);
```

## Parameters

*ppResources*

Type: [ID3D11RenderTargetView\\*\\*](#)

A pointer to an array of render-target-view interfaces. See [ID3D11RenderTargetView](#).

*Offset*

Type: [UINT](#)

The zero-based array index to get the first interface.

*Count*

Type: [UINT](#)

The number of elements in the array.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRenderTargetViewVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRenderTargetViewVariable::SetRenderTarget method

Article • 03/15/2021

Set a render-target.

## Syntax

C++

```
HRESULT SetRenderTarget(  
    ID3D11RenderTargetView *pResource  
>;
```

## Parameters

*pResource*

Type: [ID3D11RenderTargetView\\*](#)

A pointer to a render-target-view interface. See [ID3D11RenderTargetView](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRenderTargetViewVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectRenderTargetViewVariable::SetRenderTargetArray method

Article • 03/15/2021

Set an array of render-targets.

## Syntax

C++

```
HRESULT SetRenderTargetArray(  
    ID3D11RenderTargetView **ppResources,  
    UINT             Offset,  
    UINT             Count  
) ;
```

## Parameters

*ppResources*

Type: [ID3D11RenderTargetView\\*\\*](#)

Set an array of render-target-view interfaces. See [ID3D11RenderTargetView](#).

*Offset*

Type: [UINT](#)

The zero-based array index to store the first interface.

*Count*

Type: [UINT](#)

The number of elements in the array.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectRenderTargetViewVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectSamplerVariable interface

Article • 03/15/2021

A sampler interface accesses sampler state.

## Members

The **ID3DX11EffectSamplerVariable** interface inherits from [ID3DX11EffectVariable](#). **ID3DX11EffectSamplerVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectSamplerVariable** interface has these methods.

Method	Description
<a href="#">GetBackingStore</a>	Get a pointer to a variable that contains sampler state.
<a href="#">GetSampler</a>	Get a pointer to a sampler interface.
<a href="#">SetSampler</a>	Set sampler state.
<a href="#">UndoSetSampler</a>	Revert a previously set sampler state.

## Remarks

An [ID3DX11EffectVariable](#) interface is created when an effect is read into memory.

Effect variables are saved in memory in the backing store; when a technique is applied, the values in the backing store are copied to the device. You can use either of these methods to return state.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectSamplerVariable::GetBackingStore method

Article • 03/15/2021

Get a pointer to a variable that contains sampler state.

## Syntax

C++

```
HRESULT GetBackingStore(
    UINT             Index,
    D3D11_SAMPLER_DESC *pSamplerDesc
);
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of sampler descriptions. If there is only one sampler variable in the effect, use 0.

*pSamplerDesc*

Type: [D3D11\\_SAMPLER\\_DESC\\*](#)

A pointer to a sampler description (see [D3D11\\_SAMPLER\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectSamplerVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectSamplerVariable::GetSampler method

Article • 03/15/2021

Get a pointer to a sampler interface.

## Syntax

C++

```
HRESULT GetSampler(  
    UINT             Index,  
    ID3D11SamplerState **ppSampler  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of sampler interfaces. If there is only one sampler interface, use 0.

*ppSampler*

Type: [ID3D11SamplerState\\*\\*](#)

The address of a pointer to a sampler interface (see [ID3D11SamplerState](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectSamplerVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectSamplerVariable::SetSampler method

Article • 03/15/2021

Set sampler state.

## Syntax

C++

```
HRESULT SetSampler(  
    UINT             Index,  
    ID3D11SamplerState *pSampler  
>;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of sampler interfaces. If there is only one sampler interface, use 0.

*pSampler*

Type: [ID3D11SamplerState\\*](#)

Pointer to an [ID3D11SamplerState](#) interface containing the sampler state.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectSamplerVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectSamplerVariable::UndoSetSampler method

Article • 03/15/2021

Revert a previously set sampler state.

## Syntax

C++

```
HRESULT UndoSetSampler(  
    UINT Index  
>;
```

## Parameters

*Index*

Type: [UINT](#)

Index into an array of sampler interfaces. If there is only one sampler interface, use 0.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectSamplerVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable interface

Article • 03/15/2021

An effect-scalar-variable interface accesses scalar values.

## Members

The **ID3DX11EffectScalarVariable** interface inherits from [ID3DX11EffectVariable](#).  
**ID3DX11EffectScalarVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectScalarVariable** interface has these methods.

Method	Description
<a href="#">GetBool</a>	Get a boolean variable.
<a href="#">GetBoolArray</a>	Get an array of boolean variables.
<a href="#">GetFloat</a>	Get a floating-point variable.
<a href="#">GetFloatArray</a>	Get an array of floating-point variables.
<a href="#">GetInt</a>	Get an integer variable.
<a href="#">GetIntArray</a>	Get an array of integer variables.
<a href="#">SetBool</a>	Set a boolean variable.
<a href="#">SetBoolArray</a>	Set an array of boolean variables.
<a href="#">SetFloat</a>	Set a floating-point variable.
<a href="#">SetFloatArray</a>	Set an array of floating-point variables.
<a href="#">SetInt</a>	Set an integer variable.
<a href="#">SetIntArray</a>	Set an array of integer variables.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::GetBool method

Article • 03/15/2021

Get a boolean variable.

## Syntax

C++

```
HRESULT GetBool(  
    BOOL *pValue  
) ;
```

## Parameters

*pValue*

Type: **BOOL\***

A pointer to the variable.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::GetBoolArray method

Article • 03/15/2021

Get an array of boolean variables.

## Syntax

C++

```
HRESULT GetBoolArray(  
    BOOL *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: [BOOL\\*](#)

A pointer to the start of the data to set.

*Offset*

Type: [UINT](#)

Must be set to 0; this is reserved for future use.

*Count*

Type: [UINT](#)

The number of array elements to set.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::GetFloat method

Article • 03/15/2021

Get a floating-point variable.

## Syntax

C++

```
HRESULT GetFloat(  
    float *pValue  
) ;
```

## Parameters

*pValue*

Type: **float\***

A pointer to the variable.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::GetFloatArray method

Article • 03/15/2021

Get an array of floating-point variables.

## Syntax

C++

```
HRESULT GetFloatArray(  
    float *pData,  
    UINT  Offset,  
    UINT  Count  
) ;
```

## Parameters

*pData*

Type: **float\***

A pointer to the start of the data to set.

*Offset*

Type: **UINT**

Must be set to 0; this is reserved for future use.

*Count*

Type: **UINT**

The number of array elements to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::GetInt method

Article • 03/15/2021

Get an integer variable.

## Syntax

C++

```
HRESULT GetInt(  
    int *pValue  
) ;
```

## Parameters

*pValue*

Type: [int\\*](#)

A pointer to the variable.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::GetIntArray method

Article • 03/15/2021

Get an array of integer variables.

## Syntax

C++

```
HRESULT GetIntArray(  
    int *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: [int\\*](#)

A pointer to the start of the data to set.

*Offset*

Type: [UINT](#)

Must be set to 0; this is reserved for future use.

*Count*

Type: [UINT](#)

The number of array elements to set.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::SetBool method

Article • 03/15/2021

Set a boolean variable.

## Syntax

C++

```
HRESULT SetBool(  
    BOOL Value  
>;
```

## Parameters

*Value*

Type: [BOOL](#)

A pointer to the variable.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::SetBoolArray method

Article • 03/15/2021

Set an array of boolean variables.

## Syntax

C++

```
HRESULT SetBoolArray(  
    BOOL *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: [BOOL\\*](#)

A pointer to the start of the data to set.

*Offset*

Type: [UINT](#)

Must be set to 0; this is reserved for future use.

*Count*

Type: [UINT](#)

The number of array elements to set.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::SetFloat method

Article • 03/15/2021

Set a floating-point variable.

## Syntax

C++

```
HRESULT SetFloat(  
    float Value  
);
```

## Parameters

*Value*

Type: **float**

A pointer to the variable.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::SetFloatArray method

Article • 03/15/2021

Set an array of floating-point variables.

## Syntax

C++

```
HRESULT SetFloatArray(  
    float *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: **float\***

A pointer to the start of the data to set.

*Offset*

Type: **UINT**

Must be set to 0; this is reserved for future use.

*Count*

Type: **UINT**

The number of array elements to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::SetInt method

Article • 03/15/2021

Set an integer variable.

## Syntax

C++

```
HRESULT SetInt(  
    int Value  
);
```

## Parameters

*Value*

Type: [int](#)

A pointer to the variable.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectScalarVariable::SetIntArray method

Article • 03/15/2021

Set an array of integer variables.

## Syntax

C++

```
HRESULT SetIntArray(  
    int *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: [int\\*](#)

A pointer to the start of the data to set.

*Offset*

Type: [UINT](#)

Must be set to 0; this is reserved for future use.

*Count*

Type: [UINT](#)

The number of array elements to set.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectScalarVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderResourceVariable interface

Article • 03/15/2021

A shader-resource interface accesses a shader resource.

## Members

The **ID3DX11EffectShaderResourceVariable** interface inherits from [ID3DX11EffectVariable](#). **ID3DX11EffectShaderResourceVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectShaderResourceVariable** interface has these methods.

Method	Description
<a href="#">GetResource</a>	Get a shader resource.
<a href="#">GetResourceArray</a>	Get an array of shader resources.
<a href="#">SetResource</a>	Set a shader resource.
<a href="#">SetResourceArray</a>	Set an array of shader resources.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderResourceVariable::GetResource method

Article • 03/15/2021

Get a shader resource.

## Syntax

C++

```
HRESULT GetResource(  
    ID3D11ShaderResourceView **ppResource  
);
```

## Parameters

*ppResource*

Type: [ID3D11ShaderResourceView\\*\\*](#)

The address of a pointer to a shader-resource-view interface. See [ID3D11ShaderResourceView](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderResourceVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderResourceVariable::GetResourceArray method

Article • 03/15/2021

Get an array of shader resources.

## Syntax

C++

```
HRESULT GetResourceArray(
    ID3D11ShaderResourceView **ppResources,
    UINT                 Offset,
    UINT                 Count
);
```

## Parameters

*ppResources*

Type: [ID3D11ShaderResourceView\\*\\*](#)

The address of an array of shader-resource-view interfaces. See [ID3D11ShaderResourceView](#).

*Offset*

Type: [UINT](#)

The zero-based array index to get the first interface.

*Count*

Type: [UINT](#)

The number of elements in the array.

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderResourceVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderResourceVariable::SetResource method

Article • 03/15/2021

Set a shader resource.

## Syntax

C++

```
HRESULT SetResource(  
    ID3D11ShaderResourceView *pResource  
);
```

## Parameters

*pResource*

Type: [ID3D11ShaderResourceView\\*](#)

The address of a pointer to a shader-resource-view interface. See [ID3D11ShaderResourceView](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderResourceVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderResourceVariable::SetResourceArray method

Article • 03/15/2021

Set an array of shader resources.

## Syntax

C++

```
HRESULT SetResourceArray(  
    ID3D11ShaderResourceView **ppResources,  
    UINT                 Offset,  
    UINT                 Count  
>;
```

## Parameters

*ppResources*

Type: [ID3D11ShaderResourceView\\*\\*](#)

The address of an array of shader-resource-view interfaces. See [ID3D11ShaderResourceView](#).

*Offset*

Type: [UINT](#)

The zero-based array index to get the first interface.

*Count*

Type: [UINT](#)

The number of elements in the array.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderResourceVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable interface

Article • 03/15/2021

A shader-variable interface accesses a shader variable.

## Members

The **ID3DX11EffectShaderVariable** interface inherits from [ID3DX11EffectVariable](#).  
**ID3DX11EffectShaderVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectShaderVariable** interface has these methods.

Method	Description
<a href="#">GetComputeShader</a>	Get a compute shader.
<a href="#">GetDomainShader</a>	Get a domain shader.
<a href="#">GetGeometryShader</a>	Get a geometry shader.
<a href="#">GetHullShader</a>	Get a hull shader.
<a href="#">GetInputSignatureElementDesc</a>	Get an input-signature description.
<a href="#">GetOutputSignatureElementDesc</a>	Get an output-signature description.
<a href="#">GetPatchConstantSignatureElementDesc</a>	Get a patch constant signature description.
<a href="#">GetPixelShader</a>	Get a pixel shader.
<a href="#">GetShaderDesc</a>	Get a shader description.
<a href="#">GetVertexShader</a>	Get a vertex shader.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about

using Effects 11 source, see Differences Between Effects 10 and Effects 11.

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetComputeShader method

Article • 03/15/2021

Get a compute shader.

## Syntax

C++

```
HRESULT GetComputeShader(  
    UINT             ShaderIndex,  
    ID3D11ComputeShader **ppPS  
) ;
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

Index of the compute shader.

*ppPS*

Type: [ID3D11ComputeShader\\*\\*](#)

Pointer to an [ID3D11ComputeShader](#) pointer that will be set to the compute shader on return.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetDomainShader method

Article • 03/15/2021

Get a domain shader.

## Syntax

C++

```
HRESULT GetDomainShader(  
    UINT             ShaderIndex,  
    ID3D11DomainShader **ppPS  
) ;
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

Index of the domain shader.

*ppPS*

Type: [ID3D11DomainShader\\*\\*](#)

Pointer to an [ID3D11DomainShader](#) pointer that will be set to the domain shader on return.

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetGeometryShader method

Article • 03/15/2021

Get a geometry shader.

## Syntax

C++

```
HRESULT GetGeometryShader(  
    UINT             ShaderIndex,  
    ID3D11GeometryShader **ppGS  
) ;
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

A zero-based index.

*ppGS*

Type: [ID3D11GeometryShader\\*\\*](#)

A pointer to an [ID3D11GeometryShader](#) pointer that will be set to the geometry shader on return.

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetHullShader method

Article • 03/15/2021

Get a hull shader.

## Syntax

C++

```
HRESULT GetHullShader(  
    UINT             ShaderIndex,  
    ID3D11HullShader **ppPS  
) ;
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

Index of the shader.

*ppPS*

Type: [ID3D11HullShader\\*\\*](#)

A pointer to an [ID3D11HullShader](#) pointer that will be set to the hull shader on return.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetInputSignatureElementDesc method

Article • 03/15/2021

Get an input-signature description.

## Syntax

C++

```
HRESULT GetInputSignatureElementDesc(
    UINT             ShaderIndex,
    UINT             Element,
    D3D11_SIGNATURE_PARAMETER_DESC *pDesc
);
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

A zero-based shader index.

*Element*

Type: [UINT](#)

A zero-based shader-element index.

*pDesc*

Type: [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC\\*](#)

A pointer to a parameter description (see [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

An effect contains one or more shaders; each shader has an input and output signature; each signature contains one or more elements (or parameters). The shader index identifies the shader and the element index identifies the element (or parameter) in the shader signature.

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetOutputSignatureElementDesc method

Article • 03/15/2021

Get an output-signature description.

## Syntax

C++

```
HRESULT GetOutputSignatureElementDesc(
    UINT             ShaderIndex,
    UINT             Element,
    D3D11_SIGNATURE_PARAMETER_DESC *pDesc
);
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

A zero-based shader index.

*Element*

Type: [UINT](#)

A zero-based element index.

*pDesc*

Type: [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC\\*](#)

A pointer to a parameter description (see [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

An effect contains one or more shaders; each shader has an input and output signature; each signature contains one or more elements (or parameters). The shader index identifies the shader and the element index identifies the element (or parameter) in the shader signature.

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetPatchConstantSignatureElementDesc method

Article • 03/15/2021

Get a patch constant signature description.

## Syntax

C++

```
HRESULT GetPatchConstantSignatureElementDesc(
    UINT             ShaderIndex,
    UINT             Element,
    D3D11_SIGNATURE_PARAMETER_DESC *pDesc
);
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

A zero-based shader index.

*Element*

Type: [UINT](#)

A zero-based element index.

*pDesc*

Type: [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC\\*](#)

A pointer to a parameter description (see [D3D11\\_SIGNATURE\\_PARAMETER\\_DESC](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetPixelShader method

Article • 03/15/2021

Get a pixel shader.

## Syntax

C++

```
HRESULT GetPixelShader(  
    UINT             ShaderIndex,  
    ID3D11PixelShader **ppPS  
) ;
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

A zero-based index.

*ppPS*

Type: [ID3D11PixelShader\\*\\*](#)

A pointer to an [ID3D11PixelShader](#) pointer that will be set to the pixel shader on return.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetShaderDesc method

Article • 03/15/2021

Get a shader description.

## Syntax

C++

```
HRESULT GetShaderDesc(  
    UINT ShaderIndex,  
    D3DX11_EFFECT_SHADER_DESC *pDesc  
)
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

A zero-based index.

*pDesc*

Type: [D3DX11\\_EFFECT\\_SHADER\\_DESC\\*](#)

A pointer to a shader description (see [D3DX11\\_EFFECT\\_SHADER\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectShaderVariable::GetVertexShader method

Article • 03/15/2021

Get a vertex shader.

## Syntax

C++

```
HRESULT GetVertexShader(  
    UINT             ShaderIndex,  
    ID3D11VertexShader **ppVS  
) ;
```

## Parameters

*ShaderIndex*

Type: [UINT](#)

A zero-based index.

*ppVS*

Type: [ID3D11VertexShader\\*\\*](#)

A pointer to an [ID3D11VertexShader](#) pointer that will be set to the vertex shader on return.

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectShaderVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectStringVariable interface

Article • 03/15/2021

A string-variable interface accesses a string variable.

## Members

The **ID3DX11EffectStringVariable** interface inherits from [ID3DX11EffectVariable](#).  
**ID3DX11EffectStringVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectStringVariable** interface has these methods.

Method	Description
<a href="#">GetString</a>	Get the string.
<a href="#">GetStringArray</a>	Get an array of strings.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectStringVariable::GetString method

Article • 03/15/2021

Get the string.

## Syntax

C++

```
HRESULT GetString(  
    LPCSTR *ppString  
);
```

## Parameters

*ppString*

Type: [LPCSTR\\*](#)

A pointer to the string.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectStringVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectStringVariable::GetStringArray method

Article • 03/15/2021

Get an array of strings.

## Syntax

C++

```
HRESULT GetStringArray(  
    LPCSTR *ppStrings,  
    UINT    Offset,  
    UINT    Count  
) ;
```

## Parameters

*ppStrings*

Type: [LPCSTR\\*](#)

A pointer to the first string in the array.

*Offset*

Type: [UINT](#)

The offset (in number of strings) between the start of the array and the first string to get.

*Count*

Type: [UINT](#)

The number of strings in the returned array.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectStringVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectTechnique interface

Article • 03/15/2021

An **ID3DX11EffectTechnique** interface is a collection of passes.

The lifetime of an **ID3DX11EffectTechnique** object is equal to the lifetime of its parent **ID3DX11Effect** object.

- [Methods](#)

## Methods

The **ID3DX11EffectTechnique** interface has these methods.

Method	Description
<a href="#">ComputeStateBlockMask</a>	Compute a state-block mask to allow/prevent state changes.
<a href="#">GetAnnotationByIndex</a>	Get an annotation by index.
<a href="#">GetAnnotationByName</a>	Get an annotation by name.
<a href="#">GetDesc</a>	Get a technique description.
<a href="#">GetPassByIndex</a>	Get a pass by index.
<a href="#">GetPassByName</a>	Get a pass by name.
<a href="#">IsValid</a>	Test a technique to see if it contains valid syntax.

## Remarks

An effect contains one or more techniques; each technique contains one or more passes; each pass contains state assignments.

To get an effect-technique interface, call a method such as

[\*\*ID3DX11Effect::GetTechniqueByName\*\*](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectTechnique::ComputeStateBlockMask method

Article • 03/15/2021

Compute a state-block mask to allow/prevent state changes.

## Syntax

C++

```
HRESULT ComputeStateBlockMask(  
    D3DX11_STATE_BLOCK_MASK *pStateBlockMask  
) ;
```

## Parameters

*pStateBlockMask*

Type: [D3DX11\\_STATE\\_BLOCK\\_MASK\\*](#)

A pointer to a state-block mask (see [D3DX11\\_STATE\\_BLOCK\\_MASK](#)).

## Return value

Type: [HRESULT](#) ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectTechnique](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectTechnique::GetAnnotationByIndex method

Article • 03/15/2021

Get an annotation by index.

## Syntax

C++

```
ID3DX11EffectVariable* GetAnnotationByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

The zero-based index of the interface pointer.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

Use an annotation to attach a piece of metadata to a technique.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectTechnique](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectTechnique::GetAnnotationByName method

Article • 03/15/2021

Get an annotation by name.

## Syntax

C++

```
ID3DX11EffectVariable* GetAnnotationByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

Name of the annotation.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

Use an annotation to attach a piece of metadata to a technique.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectTechnique](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectTechnique::GetDesc method

Article • 03/15/2021

Get a technique description.

## Syntax

C++

```
HRESULT GetDesc(  
    D3DX11_TECHNIQUE_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_TECHNIQUE\\_DESC\\*](#)

A pointer to a technique description (see [D3DX11\\_TECHNIQUE\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectTechnique](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectTechnique::GetPassByIndex method

Article • 03/15/2021

Get a pass by index.

## Syntax

C++

```
ID3DX11EffectPass* GetPassByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectPass\\*](#)

A pointer to a [ID3DX11EffectPass](#).

## Remarks

A technique contains one or more passes; get a pass using a name or an index.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectTechnique](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectTechnique::GetPassByName method

Article • 03/15/2021

Get a pass by name.

## Syntax

C++

```
ID3DX11EffectPass* GetPassByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

The name of the pass.

## Return value

Type: [ID3DX11EffectPass\\*](#)

A pointer to an [ID3DX11EffectPass](#).

## Remarks

A technique contains one or more passes; get a pass using a name or an index.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectTechnique](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectTechnique::IsValid method

Article • 03/15/2021

Test a technique to see if it contains valid syntax.

## Syntax

C++

```
BOOL IsValid();
```

## Parameters

This method has no parameters.

## Return value

Type: **BOOL**

TRUE if the code syntax is valid; otherwise FALSE.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectTechnique](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectType interface

Article • 03/15/2021

The **ID3DX11EffectType** interface accesses effect variables by type.

The lifetime of an **ID3DX11EffectType** object is equal to the lifetime of its parent **ID3DX11Effect** object.

- Methods

## Methods

The **ID3DX11EffectType** interface has these methods.

Method	Description
<a href="#">GetDesc</a>	Get an effect-type description.
<a href="#">GetMemberName</a>	Get the name of a member.
<a href="#">GetMemberSemantic</a>	Get the semantic attached to a member.
<a href="#">GetMemberTypeByIndex</a>	Get a member type by index.
<a href="#">GetMemberTypeByName</a>	Get an member type by name.
<a href="#">GetMemberTypeBySemantic</a>	Get a member type by semantic.
<a href="#">IsValid</a>	Tests that the effect type is valid.

## Remarks

To get information about an effect type from an effect variable, call [ID3DX11EffectVariable::GetType](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectType::GetDesc method

Article • 03/15/2021

Get an effect-type description.

## Syntax

C++

```
HRESULT GetDesc(  
    D3DX11_EFFECT_TYPE_DESC *pDesc  
) ;
```

## Parameters

*pDesc*

Type: [D3DX11\\_EFFECT\\_TYPE\\_DESC\\*](#)

A pointer to an effect-type description. See [D3DX11\\_EFFECT\\_TYPE\\_DESC](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

The effect-variable description contains data about the name, annotations, semantic, flags and buffer offset of the effect type.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectType](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectType::GetMemberName method

Article • 03/15/2021

Get the name of a member.

## Syntax

C++

```
LPCSTR GetMemberName(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [LPCSTR](#)

The name of the member.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectType](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectType::GetMemberSemantic method

Article • 03/15/2021

Get the semantic attached to a member.

## Syntax

C++

```
LPCSTR GetMemberSemantic(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [LPCSTR](#)

A string that contains the semantic.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectType](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectType::GetMemberTypeByIndex method

Article • 03/15/2021

Get a member type by index.

## Syntax

C++

```
ID3DX11EffectType* GetMemberTypeByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectType\\*](#)

A pointer to an [ID3DX11EffectType](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectType](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectType::GetMemberTypeByName method

Article • 07/18/2024

Get a member type by name.

## Syntax

C++

```
ID3DX11EffectType* GetMemberTypeByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

A member's name.

## Return value

Type: [ID3DX11EffectType\\*](#)

A pointer to an [ID3DX11EffectType](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Expand table

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectType](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ID3DX11EffectType::GetMemberTypeBySemantic method

Article • 03/15/2021

Get a member type by semantic.

## Syntax

C++

```
ID3DX11EffectType* GetMemberTypeBySemantic(  
    LPCSTR Semantic  
);
```

## Parameters

*Semantic*

Type: [LPCSTR](#)

A semantic.

## Return value

Type: [ID3DX11EffectType\\*](#)

A pointer to an [ID3DX11EffectType](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectType](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectType::IsValid method

Article • 03/15/2021

Tests that the effect type is valid.

## Syntax

C++

```
BOOL IsValid();
```

## Parameters

This method has no parameters.

## Return value

Type: **BOOL**

TRUE if it is valid; otherwise FALSE.

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectType](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectUnorderedAccessViewVariable interface

Article • 03/15/2021

Accesses an unordered access view.

## Members

The **ID3DX11EffectUnorderedAccessViewVariable** interface inherits from [ID3DX11EffectVariable](#). **ID3DX11EffectUnorderedAccessViewVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectUnorderedAccessViewVariable** interface has these methods.

Method	Description
<a href="#">GetUnorderedAccessView</a>	Get an unordered-access-view.
<a href="#">GetUnorderedAccessViewArray</a>	Get an array of unordered-access-views.
<a href="#">SetUnorderedAccessView</a>	Set an unordered-access-view.
<a href="#">SetUnorderedAccessViewArray</a>	Set an array of unordered-access-views.

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectUnorderedAccessViewVariable::GetUnorderedAccessView method

Article • 03/15/2021

Get an unordered-access-view.

## Syntax

C++

```
HRESULT GetUnorderedAccessView(
    ID3D11UnorderedAccessView **ppResource
);
```

## Parameters

*ppResource*

Type: [ID3D11UnorderedAccessView\\*\\*](#)

Pointer to an [ID3D11UnorderedAccessView](#) pointer that will be set on return.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectUnorderedAccessViewVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectUnorderedAccessViewVariable::GetUnorderedAccessViewArray method

Article • 03/15/2021

Get an array of unordered-access-views.

## Syntax

C++

```
HRESULT GetUnorderedAccessViewArray(
    ID3D11UnorderedAccessView **ppResources,
    UINT                 Offset,
    UINT                 Count
);
```

## Parameters

*ppResources*

Type: [ID3D11UnorderedAccessView\\*\\*](#)

Pointer to an [ID3D11UnorderedAccessView](#) pointer that will be set to the UAV array on return.

*Offset*

Type: [UINT](#)

Index of the first interface.

*Count*

Type: [UINT](#)

Number of elements in the array.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectUnorderedAccessViewVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectUnorderedAccessViewVariable::SetUnorderedAccessView method

Article • 03/15/2021

Set an unordered-access-view.

## Syntax

C++

```
HRESULT SetUnorderedAccessView(  
    ID3D11UnorderedAccessView *pResource  
>;
```

## Parameters

*pResource*

Type: [ID3D11UnorderedAccessView\\*](#)

Pointer to an [ID3D11UnorderedAccessView](#).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectUnorderedAccessViewVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectUnorderedAccessViewVariable::SetUnorderedAccessViewArray method

Article • 03/15/2021

Set an array of unordered-access-views.

## Syntax

C++

```
HRESULT SetUnorderedAccessViewArray(
    ID3D11UnorderedAccessView **ppResources,
    UINT                 Offset,
    UINT                 Count
);
```

## Parameters

*ppResources*

Type: [ID3D11UnorderedAccessView\\*\\*](#)

An array of [ID3D11UnorderedAccessView](#) pointers.

*Offset*

Type: [UINT](#)

Index of the first unordered-access-view.

*Count*

Type: [UINT](#)

Number of elements in the array.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectUnorderedAccessViewVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable interface

Article • 03/15/2021

The **ID3DX11EffectVariable** interface is the base class for all effect variables.

The lifetime of an **ID3DX11EffectVariable** object is equal to the lifetime of its parent **ID3DX11Effect** object.

- [Methods](#)

## Methods

The **ID3DX11EffectVariable** interface has these methods.

Method	Description
<a href="#">AsBlend</a>	Get a effect-blend variable.
<a href="#">AsClassInstance</a>	Get a class-instance variable.
<a href="#">AsConstantBuffer</a>	Get a constant buffer.
<a href="#">AsDepthStencil</a>	Get a depth-stencil variable.
<a href="#">AsDepthStencilView</a>	Get a depth-stencil-view variable.
<a href="#">AsInterface</a>	Get an interface variable.
<a href="#">AsMatrix</a>	Get a matrix variable.
<a href="#">AsRasterizer</a>	Get a rasterizer variable.
<a href="#">AsRenderTargetView</a>	Get a render-target-view variable.
<a href="#">AsSampler</a>	Get a sampler variable.
<a href="#">AsScalar</a>	Get a scalar variable.
<a href="#">AsShader</a>	Get a shader variable.
<a href="#">AsShaderResource</a>	Get a shader-resource variable.
<a href="#">AsString</a>	Get a string variable.
<a href="#">AsUnorderedAccessView</a>	Get an unordered-access-view variable.
<a href="#">AsVector</a>	Get a vector variable.
<a href="#">GetAnnotationByIndex</a>	Get an annotation by index.

Method	Description
<a href="#">GetAnnotationByName</a>	Get an annotation by name.
<a href="#">GetDesc</a>	Get a description.
<a href="#">GetElement</a>	Get an array element.
<a href="#">GetMemberByIndex</a>	Get a structure member by index.
<a href="#">GetMemberByName</a>	Get a structure member by name.
<a href="#">GetMemberBySemantic</a>	Get a structure member by semantic.
<a href="#">GetParentConstantBuffer</a>	Get a constant buffer.
<a href="#">GetRawValue</a>	Get data.
<a href="#">GetType</a>	Get type information.
<a href="#">IsValid</a>	Compare the data type with the data stored.
<a href="#">SetRawValue</a>	Set data.

## Remarks

 Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsBlend method

Article • 03/15/2021

Get a effect-blend variable.

## Syntax

C++

```
ID3DX11EffectBlendVariable* AsBlend();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectBlendVariable\\*](#)

A pointer to an effect blend variable. See [ID3DX11EffectBlendVariable](#).

## Remarks

AsBlend returns a version of the effect variable that has been specialized to an effect-blend variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain effect-blend data.

Applications can test the returned object for validity by calling [IsValid](#).

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsClassInstance method

Article • 03/15/2021

Get a class-instance variable.

## Syntax

C++

```
ID3DX11EffectClassInstanceVariable* AsClassInstance();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectClassInstanceVariable\\*](#)

A pointer to class-instance variable. See [ID3DX11EffectClassInstanceVariable](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsConstantBuffer method

Article • 03/15/2021

Get a constant buffer.

## Syntax

C++

```
ID3DX11EffectConstantBuffer* AsConstantBuffer();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectConstantBuffer\\*](#)

A pointer to a constant buffer. See [ID3DX11EffectConstantBuffer](#).

## Remarks

AsConstantBuffer returns a version of the effect variable that has been specialized to a constant buffer. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain constant buffer data.

Applications can test the returned object for validity by calling [IsValid](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsDepthStencil method

Article • 03/15/2021

Get a depth-stencil variable.

## Syntax

C++

```
ID3DX11EffectDepthStencilVariable* AsDepthStencil();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectDepthStencilVariable\\*](#)

A pointer to a depth-stencil variable. See [ID3DX11EffectDepthStencilVariable](#).

## Remarks

AsDepthStencil returns a version of the effect variable that has been specialized to a depth-stencil variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain depth-stencil data.

Applications can test the returned object for validity by calling [IsValid](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsDepthStencilView method

Article • 03/15/2021

Get a depth-stencil-view variable.

## Syntax

C++

```
ID3DX11EffectDepthStencilViewVariable* AsDepthStencilView();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectDepthStencilViewVariable\\*](#)

A pointer to a depth-stencil-view variable. See [ID3DX11EffectDepthStencilViewVariable](#).

## Remarks

This method returns a version of the effect variable that has been specialized to a depth-stencil-view variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain depth-stencil-view data.

Applications can test the returned object for validity by calling [IsValid](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsInterface method

Article • 03/15/2021

Get an interface variable.

## Syntax

C++

```
ID3DX11EffectInterfaceVariable* AsInterface();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectInterfaceVariable\\*](#)

A pointer to an interface variable. See [ID3DX11EffectInterfaceVariable](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsMatrix method

Article • 03/15/2021

Get a matrix variable.

## Syntax

C++

```
ID3DX11EffectMatrixVariable* AsMatrix();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectMatrixVariable\\*](#)

A pointer to a matrix variable. See [ID3DX11EffectMatrixVariable](#).

## Remarks

AsMatrix returns a version of the effect variable that has been specialized to a matrix variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain matrix data.

Applications can test the returned object for validity by calling [IsValid](#).

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsRasterizer method

Article • 03/15/2021

Get a rasterizer variable.

## Syntax

C++

```
ID3DX11EffectRasterizerVariable* AsRasterizer();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectRasterizerVariable\\*](#)

A pointer to a rasterizer variable. See [ID3DX11EffectRasterizerVariable](#).

## Remarks

AsRasterizer returns a version of the effect variable that has been specialized to a rasterizer variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain rasterizer data.

Applications can test the returned object for validity by calling [IsValid](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsRenderTargetView method

Article • 03/15/2021

Get a render-target-view variable.

## Syntax

C++

```
ID3DX11EffectRenderTargetViewVariable* AsRenderTargetView();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectRenderTargetViewVariable\\*](#)

A pointer to a render-target-view variable. See  
[ID3DX11EffectRenderTargetViewVariable](#).

## Remarks

This method returns a version of the effect variable that has been specialized to a render-target-view variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain render-target-view data.

Applications can test the returned object for validity by calling [IsValid](#).

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsSampler method

Article • 03/15/2021

Get a sampler variable.

## Syntax

C++

```
ID3DX11EffectSamplerVariable* AsSampler();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectSamplerVariable\\*](#)

A pointer to a sampler variable. See [ID3DX11EffectSamplerVariable](#).

## Remarks

AsSampler returns a version of the effect variable that has been specialized to a sampler variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain sampler data.

Applications can test the returned object for validity by calling [IsValid](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsScalar method

Article • 03/15/2021

Get a scalar variable.

## Syntax

C++

```
ID3DX11EffectScalarVariable* AsScalar();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectScalarVariable\\*](#)

A pointer to a scalar variable. See [ID3DX11EffectScalarVariable](#).

## Remarks

AsScalar returns a version of the effect variable that has been specialized to a scalar variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain scalar data.

Applications can test the returned object for validity by calling [IsValid](#).

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsShader method

Article • 03/15/2021

Get a shader variable.

## Syntax

C++

```
ID3DX11EffectShaderVariable* AsShader();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectShaderVariable\\*](#)

A pointer to a shader variable. See [ID3DX11EffectShaderVariable](#).

## Remarks

AsShader returns a version of the effect variable that has been specialized to a shader variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain shader data.

Applications can test the returned object for validity by calling [IsValid](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsShaderResource method

Article • 03/15/2021

Get a shader-resource variable.

## Syntax

C++

```
ID3DX11EffectShaderResourceVariable* AsShaderResource();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectShaderResourceVariable\\*](#)

A pointer to a shader-resource variable. See [ID3DX11EffectShaderResourceVariable](#).

## Remarks

AsShaderResource returns a version of the effect variable that has been specialized to a shader-resource variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain shader-resource data.

Applications can test the returned object for validity by calling [IsValid](#).

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsString method

Article • 03/15/2021

Get a string variable.

## Syntax

C++

```
ID3DX11EffectStringVariable* AsString();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectStringVariable\\*](#)

A pointer to a string variable. See [ID3DX11EffectStringVariable](#).

## Remarks

AsString returns a version of the effect variable that has been specialized to a string variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain string data.

Applications can test the returned object for validity by calling [IsValid](#).

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsUnorderedAccessView method

Article • 03/15/2021

Get an unordered-access-view variable.

## Syntax

C++

```
ID3DX11EffectUnorderedAccessViewVariable* AsUnorderedAccessView();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectUnorderedAccessViewVariable\\*](#)

A pointer to an unordered-access-view variable. See [ID3DX11EffectUnorderedAccessViewVariable](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

Requirement	Value
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::AsVector method

Article • 03/15/2021

Get a vector variable.

## Syntax

C++

```
ID3DX11EffectVectorVariable* AsVector();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectVectorVariable\\*](#)

A pointer to a vector variable. See [ID3DX11EffectVectorVariable](#).

## Remarks

AsVector returns a version of the effect variable that has been specialized to a vector variable. Similar to a cast, this specialization will return an invalid object if the effect variable does not contain vector data.

Applications can test the returned object for validity by calling [IsValid](#).

ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetAnnotationByIndex method

Article • 03/15/2021

Get an annotation by index.

## Syntax

C++

```
ID3DX11EffectVariable* GetAnnotationByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

Annoations can be attached to a technique, a pass, or a global variable.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetAnnotationByName method

Article • 03/15/2021

Get an annotation by name.

## Syntax

C++

```
ID3DX11EffectVariable* GetAnnotationByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

The annotation name.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#). Note that if the annotation is not found the [ID3DX11EffectVariable](#) returned will be empty. The [ID3DX11EffectVariable::IsValid](#) method should be called to determine whether the annotation was found.

## Remarks

Annoations can be attached to a technique, a pass, or a global variable.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetDesc method

Article • 03/15/2021

Get a description.

## Syntax

C++

```
HRESULT GetDesc(  
    D3DX11_EFFECT_VARIABLE_DESC *pDesc  
)
```

## Parameters

*pDesc*

Type: [D3DX11\\_EFFECT\\_VARIABLE\\_DESC\\*](#)

A pointer to an effect-variable description (see [D3DX11\\_EFFECT\\_VARIABLE\\_DESC](#)).

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
-------------	-------

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetElement method

Article • 03/15/2021

Get an array element.

## Syntax

C++

```
ID3DX11EffectVariable* GetElement(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index; otherwise 0.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

If the effect variable is an array, use this method to return one of the elements.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetMemberByIndex method

Article • 03/15/2021

Get a structure member by index.

## Syntax

C++

```
ID3DX11EffectVariable* GetMemberByIndex(  
    UINT Index  
) ;
```

## Parameters

*Index*

Type: [UINT](#)

A zero-based index.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

If the effect variable is an structure, use this method to look up a member by index.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetMemberByName method

Article • 03/15/2021

Get a structure member by name.

## Syntax

C++

```
ID3DX11EffectVariable* GetMemberByName(  
    LPCSTR Name  
) ;
```

## Parameters

*Name*

Type: [LPCSTR](#)

Member name.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

If the effect variable is an structure, use this method to look up a member by name.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetMemberBySemantic method

Article • 03/15/2021

Get a structure member by semantic.

## Syntax

C++

```
ID3DX11EffectVariable* GetMemberBySemantic(  
    LPCSTR Semantic  
);
```

## Parameters

*Semantic*

Type: [LPCSTR](#)

The semantic.

## Return value

Type: [ID3DX11EffectVariable\\*](#)

A pointer to an [ID3DX11EffectVariable](#).

## Remarks

If the effect variable is an structure, use this method to look up a member by attached semantic.

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetParentConstantBuffer method

Article • 03/15/2021

Get a constant buffer.

## Syntax

C++

```
ID3DX11EffectConstantBuffer* GetParentConstantBuffer();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectConstantBuffer\\*](#)

A pointer to a [ID3DX11EffectConstantBuffer](#).

## Remarks

Effect variables are read-from or written-to a constant buffer.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

Requirement	Value
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetRawValue method

Article • 03/15/2021

Get data.

## Syntax

C++

```
HRESULT GetRawValue(  
    void *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: **void\***

A pointer to the variable.

*Offset*

Type: **UINT**

The offset (in bytes) from the beginning of the pointer to the data.

*Count*

Type: **UINT**

The number of bytes to get.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

This method does no conversion or type checking; it is therefore a very quick way to access array items.

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

## Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::GetType method

Article • 03/15/2021

Get type information.

## Syntax

C++

```
ID3DX11EffectType* GetType();
```

## Parameters

This method has no parameters.

## Return value

Type: [ID3DX11EffectType\\*](#)

A pointer to an [ID3DX11EffectType](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::IsValid method

Article • 03/15/2021

Compare the data type with the data stored.

## Syntax

C++

```
BOOL IsValid();
```

## Parameters

This method has no parameters.

## Return value

Type: **BOOL**

TRUE if the syntax is valid; otherwise FALSE.

## Remarks

This method checks that the data type matches the data stored after casting one interface to another (using any of the As methods).

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

Requirement	Value
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVariable::SetRawValue method

Article • 03/15/2021

Set data.

## Syntax

C++

```
HRESULT SetRawValue(  
    void *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: **void\***

A pointer to the variable.

*Offset*

Type: **UINT**

The offset (in bytes) from the beginning of the pointer to the data.

*Count*

Type: **UINT**

The number of bytes to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

This method does no conversion or type checking; it is therefore a very quick way to access array items.

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

## Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable interface

Article • 03/15/2021

A vector-variable interface accesses a four-component vector.

## Members

The **ID3DX11EffectVectorVariable** interface inherits from [ID3DX11EffectVariable](#).

**ID3DX11EffectVectorVariable** also has these types of members:

- [Methods](#)

## Methods

The **ID3DX11EffectVectorVariable** interface has these methods.

Method	Description
<a href="#">GetBoolVector</a>	Get a four-component vector that contains boolean data.
<a href="#">GetBoolVectorArray</a>	Get an array of four-component vectors that contain boolean data.
<a href="#">GetFloatVector</a>	Get a four-component vector that contains floating-point data.
<a href="#">GetFloatVectorArray</a>	Get an array of four-component vectors that contain floating-point data.
<a href="#">GetIntVector</a>	Get a four-component vector that contains integer data.
<a href="#">GetIntVectorArray</a>	Get an array of four-component vectors that contain integer data.
<a href="#">SetBoolVector</a>	Set a four-component vector that contains boolean data.
<a href="#">SetBoolVectorArray</a>	Set an array of four-component vectors that contain boolean data.
<a href="#">SetFloatVector</a>	Set a four-component vector that contains floating-point data.
<a href="#">SetFloatVectorArray</a>	Set an array of four-component vectors that contain floating-point data.
<a href="#">SetIntVector</a>	Set a four-component vector that contains integer data.
<a href="#">SetIntVectorArray</a>	Set an array of four-component vectors that contain integer data.

## Remarks

## Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVariable](#)

[Effects 11 Interfaces](#)

[D3DX Interfaces](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::GetBoolVector method

Article • 03/15/2021

Get a four-component vector that contains boolean data.

## Syntax

C++

```
HRESULT GetBoolVector(  
    BOOL *pData  
);
```

## Parameters

*pData*

Type: **BOOL\***

A pointer to the first component.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::GetBoolVectorArray method

Article • 03/15/2021

Get an array of four-component vectors that contain boolean data.

## Syntax

C++

```
HRESULT GetBoolVectorArray(  
    BOOL *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: **BOOL\***

A pointer to the start of the data to set.

*Offset*

Type: **UINT**

Must be set to 0; this is reserved for future use.

*Count*

Type: **UINT**

The number of array elements to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::GetFloatVector method

Article • 03/15/2021

Get a four-component vector that contains floating-point data.

## Syntax

C++

```
HRESULT GetFloatVector(  
    float *pData  
) ;
```

## Parameters

*pData*

Type: **float\***

A pointer to the first component.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::GetFloatVectorArray method

Article • 03/15/2021

Get an array of four-component vectors that contain floating-point data.

## Syntax

C++

```
HRESULT GetFloatVectorArray(  
    float *pData,  
    UINT  Offset,  
    UINT  Count  
)
```

## Parameters

*pData*

Type: **float\***

A pointer to the start of the data to set.

*Offset*

Type: **UINT**

Must be set to 0; this is reserved for future use.

*Count*

Type: **UINT**

The number of array elements to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::GetIntVector method

Article • 03/15/2021

Get a four-component vector that contains integer data.

## Syntax

C++

```
HRESULT GetIntVector(  
    int *pData  
) ;
```

## Parameters

*pData*

Type: [int\\*](#)

A pointer to the first component.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::GetIntVectorArray method

Article • 03/15/2021

Get an array of four-component vectors that contain integer data.

## Syntax

C++

```
HRESULT GetIntVectorArray(  
    int *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: [int\\*](#)

A pointer to the start of the data to set.

*Offset*

Type: [UINT](#)

Must be set to 0; this is reserved for future use.

*Count*

Type: [UINT](#)

The number of array elements to set.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::SetBoolVector method

Article • 03/15/2021

Set a four-component vector that contains boolean data.

## Syntax

C++

```
HRESULT SetBoolVector(  
    BOOL *pData  
) ;
```

## Parameters

*pData*

Type: **BOOL\***

A pointer to the first component.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::SetBoolVectorArray method

Article • 03/15/2021

Set an array of four-component vectors that contain boolean data.

## Syntax

C++

```
HRESULT SetBoolVectorArray(  
    BOOL *pData,  
    UINT Offset,  
    UINT Count  
)
```

## Parameters

*pData*

Type: **BOOL\***

A pointer to the start of the data to set.

*Offset*

Type: **UINT**

Must be set to 0; this is reserved for future use.

*Count*

Type: **UINT**

The number of array elements to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::SetFloatVector method

Article • 03/15/2021

Set a four-component vector that contains floating-point data.

## Syntax

C++

```
HRESULT SetFloatVector(  
    float *pData  
) ;
```

## Parameters

*pData*

Type: **float\***

A pointer to the first component.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::SetFloatVectorArray method

Article • 03/15/2021

Set an array of four-component vectors that contain floating-point data.

## Syntax

C++

```
HRESULT SetFloatVectorArray(  
    float *pData,  
    UINT Offset,  
    UINT Count  
)
```

## Parameters

*pData*

Type: **float\***

A pointer to the start of the data to set.

*Offset*

Type: **UINT**

Must be set to 0; this is reserved for future use.

*Count*

Type: **UINT**

The number of array elements to set.

## Return value

Type: **HRESULT** ↗

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::SetIntVect or method

Article • 03/15/2021

Set a four-component vector that contains integer data.

## Syntax

C++

```
HRESULT SetIntVector(  
    int *pData  
) ;
```

## Parameters

*pData*

Type: [int\\*](#)

A pointer to the first component.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

## Remarks

### ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ID3DX11EffectVectorVariable::SetIntVect orArray method

Article • 03/15/2021

Set an array of four-component vectors that contain integer data.

## Syntax

C++

```
HRESULT SetIntVectorArray(  
    int *pData,  
    UINT Offset,  
    UINT Count  
) ;
```

## Parameters

*pData*

Type: [int\\*](#)

A pointer to the start of the data to set.

*Offset*

Type: [UINT](#)

Must be set to 0; this is reserved for future use.

*Count*

Type: [UINT](#)

The number of array elements to set.

## Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 11 Return Codes](#).

# Remarks

## ⓘ Note

The DirectX SDK does not supply any compiled binaries for effects. You must use Effects 11 source to build your effects-type application. For more information about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

# Requirements

Requirement	Value
Header	D3dx11effect.h
Library	N/A (An Effects 11 library is available online as shared source.)

## See also

[ID3DX11EffectVectorVariable](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Effects 11 Functions

Article • 11/04/2020

This section contains information about the Effects 11 functions.

## In this section

Topic	Description
<a href="#">D3DX11CreateEffectFromMemory</a>	Creates an effect from a binary effect or file.

## Related topics

[Effects 11 Reference](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# D3DX11CreateEffectFromMemory function

Article • 03/15/2021

Creates an effect from a binary effect or file.

## Syntax

C++

```
HRESULT D3DX11CreateEffectFromMemory(
    void          *pData,
    SIZE_T        DataLength,
    UINT          FXFlags,
    ID3D11Device *pDevice,
    ID3DX11Effect **ppEffect
);
```

## Parameters

*pData*

Type: **void\***

Blob of compiled effect data.

*DataLength*

Type: **SIZE\_T**

Length of the data blob.

*FXFlags*

Type: **UINT**

No effect flags exist. Set to zero.

*pDevice*

Type: **ID3D11Device\***

Pointer to the **ID3D11Device** on which to create Effect resources.

*ppEffect*

Type: [ID3DX11Effect\\*\\*](#)

Address of the newly created [ID3DX11Effect](#) interface.

## Return value

Type: [HRESULT](#)

The return value is one of the values listed in [Direct3D 11 Return Codes](#).

## Remarks

 Note

You must use [Effects 11 source](#) to build your effects-type application. For more info about using Effects 11 source, see [Differences Between Effects 10 and Effects 11](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Effects 11 Structures

Article • 11/04/2020

This section contains information about the Effects 11 structures.

## In this section

Topic	Description
<a href="#">D3DX11_EFFECT_DESC</a>	Describes an effect.
<a href="#">D3DX11_EFFECT_SHADER_DESC</a>	Describes an effect shader.
<a href="#">D3DX11_EFFECT_TYPE_DESC</a>	Describes an effect-variable type.
<a href="#">D3DX11_EFFECT_VARIABLE_DESC</a>	Describes an effect variable.
<a href="#">D3DX11_GROUP_DESC</a>	Describes an effect group.
<a href="#">D3DX11_PASS_DESC</a>	Describes an effect pass, which contains pipeline state.
<a href="#">D3DX11_PASS_SHADER_DESC</a>	Describes an effect pass.
<a href="#">D3DX11_STATE_BLOCK_MASK</a>	Indicates the device state.
<a href="#">D3DX11_TECHNIQUE_DESC</a>	Describes an effect technique.

## Related topics

[Effects 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_EFFECT\_DESC structure

Article • 03/15/2021

Describes an effect.

## Syntax

C++

```
typedef struct _D3DX11_EFFECT_DESC {
    UINT ConstantBuffers;
    UINT GlobalVariables;
    UINT InterfaceVariables;
    UINT Techniques;
    UINT Groups;
} D3DX11_EFFECT_DESC;
```

## Members

### ConstantBuffers

Type: [UINT](#)

Number of constant buffers in this effect.

### GlobalVariables

Type: [UINT](#)

Number of global variables in this effect.

### InterfaceVariables

Type: [UINT](#)

Number of global interfaces in this effect.

### Techniques

Type: [UINT](#)

Number of techniques in this effect.

### Groups

Type: [UINT](#)

Number of groups in this effect.

## Remarks

D3DX11\_EFFECT\_DESC is used with [ID3DX11Effect::GetDesc](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_EFFECT\_SHADER\_DESC structure

Article • 03/15/2021

Describes an effect shader.

## Syntax

C++

```
typedef struct _D3DX11_EFFECT_SHADER_DESC {
    const BYTE *pInputSignature;
    BOOL IsInline;
    const BYTE *pBytecode;
    UINT BytecodeLength;
    LPCSTR SODcls[D3D11_SO_STREAM_COUNT];
    UINT RasterizedStream;
    UINT NumInputSignatureEntries;
    UINT NumOutputSignatureEntries;
    UINT NumPatchConstantSignatureEntries;
} D3DX11_EFFECT_SHADER_DESC;
```

## Members

### pInputSignature

Type: **const BYTE\***

Passed into CreateInputLayout. Only valid on a vertex shader or geometry shader. See [ID3D11Device::CreateInputLayout](#).

### IsInline

Type: **BOOL**

**TRUE** is the shader is defined inline; otherwise **FALSE**.

### pBytecode

Type: **const BYTE\***

Shader bytecode.

### BytecodeLength

Type: **UINT**

The length of pBytecode.

### SODcls

Type: [LPCSTR](#)

Stream out declaration string (for geometry shader with SO).

### RasterizedStream

Type: [UINT](#)

Indicates which stream is rasterized. D3D11 geometry shaders can output up to four streams of data, one of which can be rasterized.

### NumInputSignatureEntries

Type: [UINT](#)

Number of entries in the input signature.

### NumOutputSignatureEntries

Type: [UINT](#)

Number of entries in the output signature.

### NumPatchConstantSignatureEntries

Type: [UINT](#)

Number of entries in the patch constant signature.

## Remarks

D3DX11\_EFFECT\_SHADER\_DESC is used with  
[ID3DX11EffectShaderVariable::GetShaderDesc](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

## See also

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_EFFECT\_TYPE\_DESC structure

Article • 03/15/2021

Describes an effect-variable type.

## Syntax

C++

```
typedef struct _D3DX11_EFFECT_TYPE_DESC {
    LPCSTR             TypeName;
    D3D10_SHADER_VARIABLE_CLASS Class;
    D3D10_SHADER_VARIABLE_TYPE   Type;
    UINT               Elements;
    UINT               Members;
    UINT               Rows;
    UINT               Columns;
    UINT               PackedSize;
    UINT               UnpackedSize;
    UINT               Stride;
} D3DX11_EFFECT_TYPE_DESC;
```

## Members

### TypeName

Type: [LPCSTR](#)

Name of the type, for example "float4" or "MyStruct".

### Class

Type: [D3D10\\_SHADER\\_VARIABLE\\_CLASS](#)

The variable class (see [D3D10\\_SHADER\\_VARIABLE\\_CLASS](#)).

### Type

Type: [D3D10\\_SHADER\\_VARIABLE\\_TYPE](#)

The variable type (see [D3D10\\_SHADER\\_VARIABLE\\_TYPE](#)).

### Elements

Type: [UINT](#)

Number of elements in this type (0 if not an array).

## Members

Type: **UINT**

Number of members (0 if not a structure).

## Rows

Type: **UINT**

Number of rows in this type (0 if not a numeric primitive).

## Columns

Type: **UINT**

Number of columns in this type (0 if not a numeric primitive).

## PackedSize

Type: **UINT**

Number of bytes required to represent this data type, when tightly packed.

## UnpackedSize

Type: **UINT**

Number of bytes occupied by this data type, when laid out in a constant buffer.

## Stride

Type: **UINT**

Number of bytes to seek between elements, when laid out in a constant buffer.

## Remarks

D3DX11\_EFFECT\_TYPE\_DESC is used with [ID3DX11EffectType::GetDesc](#)

## Requirements

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_EFFECT\_VARIABLE\_DESC structure

Article • 03/15/2021

Describes an effect variable.

## Syntax

C++

```
typedef struct _D3DX11_EFFECT_VARIABLE_DESC {
    LPCSTR Name;
    LPCSTR Semantic;
    UINT    Flags;
    UINT    Annotations;
    UINT    BufferOffset;
    UINT    ExplicitBindPoint;
} D3DX11_EFFECT_VARIABLE_DESC;
```

## Members

### Name

Type: [LPCSTR](#)

Name of this variable, annotation, or structure member.

### Semantic

Type: [LPCSTR](#)

Semantic string of this variable or structure member (NULL for annotations or if not present).

### Flags

Type: [UINT](#)

Optional flags for effect variables.

### Annotations

Type: [UINT](#)

Number of annotations on this variable (always 0 for annotations).

### BufferOffset

Type: [UINT](#)

Offset into containing cbuffer or tbuffer (always 0 for annotations or variables not in constant buffers).

### ExplicitBindPoint

Type: [UINT](#)

Used if the variable has been explicitly bound using the register keyword. Check Flags for D3DX11\_EFFECT\_VARIABLE\_EXPLICIT\_BIND\_POINT.

## Remarks

D3DX11\_EFFECT\_VARIABLE\_DESC is used with [ID3DX11EffectVariable::GetDesc](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_GROUP\_DESC structure

Article • 03/15/2021

Describes an effect group.

## Syntax

C++

```
typedef struct _D3DX11_GROUP_DESC {
    LPCSTR Name;
    UINT Techniques;
    UINT Annotations;
} D3DX11_GROUP_DESC;
```

## Members

### Name

Type: [LPCSTR](#)

Name of this group (only **NULL** if global).

### Techniques

Type: [UINT](#)

Number of techniques contained in group.

### Annotations

Type: [UINT](#)

Number of annotations on this group.

## Remarks

D3DX11\_GROUP\_DESC is used with [ID3DX11EffectTechnique::GetDesc](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

# D3DX11\_PASS\_DESC structure

Article • 03/15/2021

Describes an effect pass, which contains pipeline state.

## Syntax

C++

```
typedef struct _D3DX11_PASS_DESC {
    LPCSTR Name;
    UINT Annotations;
    BYTE *pIAInputSignature;
    SIZE_T IAInputSignatureSize;
    UINT StencilRef;
    UINT SampleMask;
    FLOAT BlendFactor[4];
} D3DX11_PASS_DESC;
```

## Members

### Name

Type: [LPCSTR](#)

Name of this pass (NULL if not anonymous).

### Annotations

Type: [UINT](#)

Number of annotations on this pass.

### pIAInputSignature

Type: [BYTE\\*](#)

Signature from the vertex shader or geometry shader (if there is no vertex shader) or NULL if neither exists.

### IAInputSignatureSize

Type: [SIZE\\_T](#)

Singature size in bytes.

## StencilRef

Type: [UINT](#)

The stencil-reference value used in the depth-stencil state.

## SampleMask

Type: [UINT](#)

The sample mask for the blend state.

## BlendFactor

Type: [FLOAT](#)

The per-component blend factors (RGBA) for the blend state.

## Remarks

D3DX11\_PASS\_DESC is used with [ID3DX11EffectPass::GetDesc](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_PASS\_SHADER\_DESC structure

Article • 03/15/2021

Describes an effect pass.

## Syntax

C++

```
typedef struct _D3DX11_PASS_SHADER_DESC {
    ID3DX11EffectShaderVariable *pShaderVariable;
    UINT                         ShaderIndex;
} D3DX11_PASS_SHADER_DESC;
```

## Members

### pShaderVariable

Type: [ID3DX11EffectShaderVariable\\*](#)

The variable that this shader came from.

### ShaderIndex

Type: [UINT](#)

The element of pShaderVariable (if an array) or 0 if not applicable.

## Remarks

D3DX11\_PASS\_SHADER\_DESC is used with [ID3DX11EffectPass](#) Get\*ShaderDesc methods.

If this is an inline shader assignment, the returned interface will be an anonymous shader variable, which is not retrievable any other way. Its name in the variable description will be "\$Anonymous". If there is no assignment of this type in the pass block, pShaderVariable != **NULL**, but pShaderVariable->IsValid() == **FALSE**.

## Requirements

Requirement	Value
-------------	-------

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_STATE\_BLOCK\_MASK structure

Article • 03/15/2021

Indicates the device state.

## Syntax

C++

```
typedef struct _D3DX11_STATE_BLOCK_MASK {
    BYTE VS;
    BYTE
    VSSamplers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_SAMPLER_SLOT_COUNT)];
    BYTE
    VSShaderResources[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_INPUT_RESOURCE_SLOT_COUNT)];
    BYTE
    VSConstantBuffers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_CONSTANT_BUFFER_API_SLOT_COUNT)];
    BYTE VSInterfaces[D3DX11_BYTES_FROM_BITS(D3D11_SHADER_MAX_INTERFACES)];
    BYTE HS;
    BYTE
    HSSamplers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_SAMPLER_SLOT_COUNT)];
    BYTE
    HSShaderResources[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_INPUT_RESOURCE_SLOT_COUNT)];
    BYTE
    HSConstantBuffers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_CONSTANT_BUFFER_API_SLOT_COUNT)];
    BYTE HSInterfaces[D3DX11_BYTES_FROM_BITS(D3D11_SHADER_MAX_INTERFACES)];
    BYTE DS;
    BYTE
    DSSamplers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_SAMPLER_SLOT_COUNT)];
    BYTE
    DSShaderResources[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_INPUT_RESOURCE_SLOT_COUNT)];
    BYTE
    DSConstantBuffers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_CONSTANT_BUFFER_API_SLOT_COUNT)];
    BYTE DSInterfaces[D3DX11_BYTES_FROM_BITS(D3D11_SHADER_MAX_INTERFACES)];
    BYTE GS;
    BYTE
    GSSamplers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_SAMPLER_SLOT_COUNT)];
    BYTE
    GSShaderResources[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_INPUT_RESOURCE_SLOT_COUNT)];
    BYTE
    GSConstantBuffers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_CONSTANT_BUFFER_API_SLOT_COUNT)];
    BYTE GSInterfaces[D3DX11_BYTES_FROM_BITS(D3D11_SHADER_MAX_INTERFACES)];
```

```
    BYTE PS;
    BYTE
PSSamplers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_SAMPLER_SLOT_COUNT)];
    BYTE
PSShaderResources[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_INPUT_RESOURCE_SLOT_COUNT)];
    BYTE
PSConstantBuffers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_CONSTANT_BUFFER_API_SLOT_COUNT)];
    BYTE PSInterfaces[D3DX11_BYTES_FROM_BITS(D3D11_SHADER_MAX_INTERFACES)];
    BYTE PSUnorderedAccessViews;
    BYTE CS;
    BYTE
CSSamplers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_SAMPLER_SLOT_COUNT)];
    BYTE
CSShaderResources[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_INPUT_RESOURCE_SLOT_COUNT)];
    BYTE
CSConstantBuffers[D3DX11_BYTES_FROM_BITS(D3D11_COMMONSHADER_CONSTANT_BUFFER_API_SLOT_COUNT)];
    BYTE CSInterfaces[D3DX11_BYTES_FROM_BITS(D3D11_SHADER_MAX_INTERFACES)];
    BYTE CSUnorderedAccessViews;
    BYTE
IAVertexBuffers[D3DX11_BYTES_FROM_BITS(D3D11_IA_VERTEX_INPUT_RESOURCE_SLOT_COUNT)];
    BYTE IAIndexBuffer;
    BYTE IAInputLayout;
    BYTE IAPrimitiveTopology;
    BYTE OMRenderTargets;
    BYTE OMDepthStencilState;
    BYTE OMBlendState;
    BYTE RSSViewports;
    BYTE RSSScissorRects;
    BYTE RSRasterizerState;
    BYTE SOBuffers;
    BYTE Predication;
} D3DX11_STATE_BLOCK_MASK;
```

## Members

### VS

Type: **BYTE**

Boolean value indicating whether to save the vertex shader state.

### VSSamplers

Type: **BYTE**

Array of vertex-shader samplers. The array is a multi-byte bitmask where each bit represents one sampler slot.

### **VSShaderSamplers**

Type: **BYTE**

Array of vertex-shader resources. The array is a multi-byte bitmask where each bit represents one resource slot.

### **VSShaderResources**

Type: **BYTE**

Array of vertex-shader constant buffers. The array is a multi-byte bitmask where each bit represents one constant buffer slot.

### **VSConstantBuffers**

Type: **BYTE**

Array of vertex-shader interfaces. The array is a multi-byte bitmask where each bit represents one interface slot.

### **HS**

Type: **BYTE**

Boolean value indicating whether to save the hull shader state.

### **HSSamplers**

Type: **BYTE**

Array of hull-shader samplers. The array is a multi-byte bitmask where each bit represents one sampler slot.

### **HSShaderResources**

Type: **BYTE**

Array of hull-shader resources. The array is a multi-byte bitmask where each bit represents one resource slot.

### **HSConstantBuffers**

Type: **BYTE**

Array of hull-shader constant buffers. The array is a multi-byte bitmask where each bit represents one constant buffer slot.

### **HSInterfaces**

Type: **BYTE**

Array of hull-shader interfaces. The array is a multi-byte bitmask where each bit represents one interface slot.

### **DS**

Type: **BYTE**

Boolean value indicating whether to save the domain shader state.

### **DSSamplers**

Type: **BYTE**

Array of domain-shader samplers. The array is a multi-byte bitmask where each bit represents one sampler slot.

### **DSShaderResources**

Type: **BYTE**

Array of domain-shader resources. The array is a multi-byte bitmask where each bit represents one resource slot.

### **DSConstantBuffers**

Type: **BYTE**

Array of domain-shader constant buffers. The array is a multi-byte bitmask where each bit represents one buffer slot.

### **DSInterfaces**

Type: **BYTE**

Array of domain-shader interfaces. The array is a multi-byte bitmask where each bit represents one interface slot.

### **GS**

Type: **BYTE**

Boolean value indicating whether to save the geometry shader state.

### **GSSamplers**

Type: **BYTE**

Array of geometry-shader samplers. The array is a multi-byte bitmask where each bit represents one sampler slot.

### **GSShaderResources**

Type: **BYTE**

Array of geometry-shader resources. The array is a multi-byte bitmask where each bit represents one resource slot.

### **GSConstantBuffers**

Type: **BYTE**

Array of geometry-shader constant buffers. The array is a multi-byte bitmask where each bit represents one buffer slot.

### **GSIInterfaces**

Type: **BYTE**

Array of geometry-shader interfaces. The array is a multi-byte bitmask where each bit represents one interface slot.

## **PS**

Type: **BYTE**

Boolean value indicating whether to save the pixel shader state.

### **PSSamplers**

Type: **BYTE**

Array of pixel-shader samplers. The array is a multi-byte bitmask where each bit represents one sampler slot.

### **PSShaderResources**

Type: **BYTE**

Array of pixel-shader resources. The array is a multi-byte bitmask where each bit represents one resource slot.

### **PSConstantBuffers**

Type: **BYTE**

Array of pixel-shader constant buffers. The array is a multi-byte bitmask where each bit represents one constant buffer slot.

### **PSInterfaces**

Type: **BYTE**

Array of pixel-shader interfaces. The array is a multi-byte bitmask where each bit represents one interface slot.

### **PSUnorderedAccessViews**

Type: **BYTE**

Boolean value indicating whether to save the pixel shader unordered access views.

### **CS**

Type: **BYTE**

Boolean value indicating whether to save the compute shader state.

### **CSSamplers**

Type: **BYTE**

Array of compute-shader samplers. The array is a multi-byte bitmask where each bit represents one sampler slot.

### **CSShaderResources**

Type: **BYTE**

Array of compute-shader resources. The array is a multi-byte bitmask where each bit represents one resource slot.

### **CSConstantBuffers**

Type: **BYTE**

Array of compute-shader constant buffers. The array is a multi-byte bitmask where each bit represents one constant buffer slot.

### **CSInterfaces**

Type: **BYTE**

Array of compute-shader interfaces. The array is a multi-byte bitmask where each bit represents one interface slot.

### **CSUnorderedAccessViews**

Type: **BYTE**

Boolean value indicating whether to save the compute shader unordered access views.

### **IVertexAttribBuffers**

Type: **BYTE**

Array of vertex buffers. The array is a multi-byte bitmask where each bit represents one resource slot.

### **IAIndexBuffer**

Type: **BYTE**

Boolean value indicating whether to save the index buffer state.

### **IAInputLayout**

Type: **BYTE**

Boolean value indicating whether to save the input layout state.

### **IAPrimitiveTopology**

Type: **BYTE**

Boolean value indicating whether to save the primitive topology state.

### **OMRenderTargets**

Type: **BYTE**

Boolean value indicating whether to save the render targets states.

### **OMDepthStencilState**

Type: **BYTE**

Boolean value indicating whether to save the depth-stencil state.

#### **OMBlendState**

Type: **BYTE**

Boolean value indicating whether to save the blend state.

#### **RSViewports**

Type: **BYTE**

Boolean value indicating whether to save the viewports states.

#### **RSScissorRects**

Type: **BYTE**

Boolean value indicating whether to save the scissor rectangles states.

#### **RSRasterizerState**

Type: **BYTE**

Boolean value indicating whether to save the rasterizer state.

#### **SOBuffers**

Type: **BYTE**

Boolean value indicating whether to save the stream-out buffers states.

#### **Predication**

Type: **BYTE**

Boolean value indicating whether to save the predication state.

## **Remarks**

A state-block mask indicates the device states that a pass or a technique changes.

## **Requirements**

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# D3DX11\_TECHNIQUE\_DESC structure

Article • 03/15/2021

Describes an effect technique.

## Syntax

C++

```
typedef struct _D3DX11_TECHNIQUE_DESC {
    LPCSTR Name;
    UINT    Passes;
    UINT    Annotations;
} D3DX11_TECHNIQUE_DESC;
```

## Members

### Name

Type: [LPCSTR](#)

Name of this technique (NULL if not anonymous).

### Passes

Type: [UINT](#)

Number of passes contained in the technique.

### Annotations

Type: [UINT](#)

Number of annotations on this technique.

## Remarks

D3DX11\_TECHNIQUE\_DESC is used with [ID3DX11EffectTechnique::GetDesc](#).

## Requirements

Requirement	Value
Header	D3dx11effect.h

## See also

[Effects 11 Structures](#)

---

## Feedback

Was this page helpful?



Yes



No

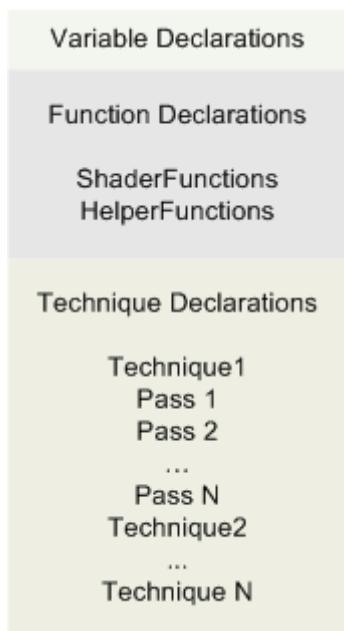
[Get help at Microsoft Q&A](#)

# Effect Format (Direct3D 11)

Article • 01/06/2021

An effect (which is often stored in a file with a .fx file extension) declares the pipeline state set by an effect. Effect state can be roughly broken down into three categories:

- [Variables](#), which are usually declared at the top of an effect.
- [Functions](#), which implement shader code, or are used as helper functions by other functions.
- [Techniques](#), which can be arranged in effect groups, and implement rendering sequences using one or more effect passes. Each pass sets one or more [state groups](#) and calls shader functions.



The preceding diagram shows the categories of effect state.

The definition of the effect binary format can be found in Binary\EffectBinaryFormat.h in the effects source code.

## In this section

Topic	Description
<a href="#">Effect Variable Syntax</a>	An effect variable is declared with the syntax described in this section.
<a href="#">Annotation Syntax</a>	An annotation is a user-defined piece of information, declared with the syntax described in this section.

Topic	Description
Effect Function Syntax	An effect function is written in HLSL and is declared with the syntax described in this section.
Effect Technique Syntax	An effect technique is declared with the syntax described in this section.
Effect State Groups	Effect states are name value pairs in the form of an expression.
Effect Group Syntax	An effect group is declared with the syntax described in this section.

## Related topics

[Effects 11 Reference](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Effect Variable Syntax (Direct3D 11)

Article • 08/19/2020

An effect variable is declared with the syntax described in this section.

## Syntax

Basic syntax:

*DataType VariableName [ : SemanticName ] < Annotations > [ = initialValue ];*

See [Variable Syntax \(DirectX HLSL\)](#) for full syntax.

Name	Description
DataType	Any <a href="#">basic</a> , <a href="#">texture</a> , unordered access view, shader or state block type.
VariableName	An ASCII string that uniquely identifies the name of the effect variable.
SemanticName	A ASCII string that denotes additional information about how a variable should be used. A semantic is an ASCII string that can be either a predefined system-value or a custom-user string.
Annotations	One or more pieces of user-supplied information (metadata) that is ignored by the effect system. For syntax, see <a href="#">Annotation Syntax (Direct3D 11)</a> .
InitialValue	The default value of the variable.

An effect variable that is declared outside of all functions, is considered global in scope; variables declared within a function are local to that function.

## Example

This example illustrates global effect numeric variables.

```
float4 g_MaterialAmbientColor;           // Material's ambient color
float4 g_MaterialDiffuseColor;           // Material's diffuse color
float3 g_LightDir[3];                   // Light's direction in world space
float4x4 g_mWorld;                     // World matrix for object
```

This example illustrates effect variables that are local to a shader function.

```
VS_OUTPUT RenderSceneVS( ... )
{
    float3 vNormalWorldSpace;
    float4 vAnimatedPos;

    // shader body
}
```

This example illustrates function parameters that have semantics.

```
VS_OUTPUT RenderSceneVS( float4 vPos : SV_POSITION,
                        float3 vNormal : NORMAL,
                        float2 vTexCoord0 : TEXCOORD0,
                        uniform int nNumLights,
                        uniform bool bTexture,
                        uniform bool bAnimate )
{
    ...
}
```

This example illustrates declaring a global texture variable.

```
Texture2D g_MeshTexture;           // Color texture for mesh
```

Sampling a texture is done with a texture sampler. To set up a sampler in an effect, see the [sampler type](#).

This example illustrates declaring global unordered access view variables.

```
RWStructuredBuffer<uint> bc : register(u2) < string name="bc"; >;
RWBuffer<uint> bRW;
struct S
{
    uint key;
    uint value;
};
AppendStructuredBuffer<S> asb : register(u5);
RWByteAddressBuffer rwbab : register(u1);
RWStructuredBuffer<uint> rwsb : register(u3);
```

```

RWTexture1D<float> rwt1d : register(u1);
RWTexture1DArray<uint> rwt1da : register(u4);
RWTexture2D<uint> rwt2d : register(u2);
RWTexture2DArray<uint> rwt2da : register(u6);
RWTexture3D<uint> rwt3d : register(u7);

    This example illustrates declaring global shader variables.

VertexShader pVS = CompileShader( vs_5_0, VS() );
HullShader pHs = NULL;
DomainShader pDS = NULL;
GeometryShader pGS = ConstructGSWithSO( CompileShader( gs_5_0, VS() ),
                                         "0:Position.xy; 1:Position.zw;
                                         2:Color.xy",
                                         "3:Texcoord.xyzw; 3:$SKIP.x;",
                                         NULL,
                                         NULL,
                                         1 );

PixelShader pPS = NULL;
ComputeShader pCS = NULL;

    This example illustrates declaring global state block variables.

BlendState myBS[2] < bool IsValid = true; >
{
{
    BlendEnable[0] = false;
},
{
    BlendEnable[0] = true;
    SrcBlendAlpha[0] = Inv_Src_Alpha;
}
};

RasterizerState myRS
{
    FillMode = Solid;
    CullMode = NONE;
    MultisampleEnable = true;
    DepthClipEnable = false;
};

DepthStencilState myDS
{
    DepthEnable = false;
    DepthWriteMask = Zero;
    DepthFunc = Less;
};

sampler mySS[2] : register(s3)
{
{
    Filter = ANISOTROPIC;
    MaxAnisotropy = 3;
},
{
    Filter = ANISOTROPIC;
    MaxAnisotropy = 4;
}
};

```

## Related topics

[Effect Format](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Annotation Syntax (Direct3D 11)

Article • 08/19/2020

An annotation is a user-defined piece of information, declared with the syntax described in this section.

```
<DataType Name = Value; ... ;>
```

## Parameters

Item	Description
<i>DataType</i>	[in] The data type, which includes any <a href="#">scalar HLSL</a> type as well as the <a href="#">string type</a> .
<i>Name</i>	[in] An ASCII string, that represents the annotation name.
<i>Value</i>	[in] The initial value of the annotation.
...	[in] Additional annotations (name-value pairs).

## Remarks

You may add more than one annotation within the angle brackets, each one separated by a semicolon. The effect framework APIs recognize annotations on global variables; all other annotations are ignored.

## Example

Here are some examples.

```
int i <int blabla=27; string blacksheep="Hello There";>;  
  
int j <int bambam=30; string blacksheep="Goodbye There";> = 5 ;  
  
float y <float y=2.3;> = 2.3, z <float y=1.3;> = 1.3 ;  
  
half w <half GlobalW = 3.62;>;  
  
float4 main(float4 pos : SV_POSITION ) : SV_POSITION  
{
```

```
pos.y = pos.x > 0 ? pos.w * 1.3 : pos.z * .032;
for (int x = i; x < j ; x++)
{
    pos.w = pos.w * pos.y + x + j - y * w;
}

return pos;
}
```

## Related topics

[Effect Format](#)

[Effect Variable Syntax](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Effect Function Syntax (Direct3D 11)

Article • 08/19/2020

An effect function is written in HLSL and is declared with the syntax described in this section.

## Syntax

*ReturnType FunctionName ( [ ArgumentList ] )*

```
{  
[ *Statements* ]  
};
```

Name	Description
ReturnType	Any <a href="#">HLSL type</a>
FunctionName	An ASCII string that uniquely identifies the name of the shader function.
ArgumentList	One or more arguments, separated by commas (see <a href="#">Function Arguments (DirectX HLSL)</a> ).
Statements	One or more statements (see <a href="#">Statements (DirectX HLSL)</a> ) that make up the body of the function. If a function is defined without a body, it is considered to be a prototype; and must be redefined with a body before use.

An effect function may be a shader or it may simply be a function called by a shader. A function is uniquely identified by its name, the types of its parameters, and the target platform; therefore, functions can be overloaded. Any valid HLSL function should fit this format; for a more detailed list of syntax for HLSL functions, see [Functions \(DirectX HLSL\)](#).

## Example

The following is an example of a pixel shader function.

```
PS_OUTPUT RenderScenePS( VS_OUTPUT In,
```

```
uniform bool bTexture )  
{  
    PS_OUTPUT Output;  
  
    // Lookup mesh texture and modulate it with diffuse  
    if( bTexture )  
        Output.RGBColor = g_MeshTexture.Sample(MeshTextureSampler,  
In.TextureUV) *  
                            In.Diffuse;  
    else  
        Output.RGBColor = In.Diffuse;  
  
    return Output;  
}
```

## Related topics

[Effect Format](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Effect Technique Syntax (Direct3D 11)

Article • 08/20/2021

An effect technique is declared with the syntax described in this section.

TechniqueVersion *TechniqueName* [ <Annotations> ]

```
{  
    pass *PassName* {[ <*Annotations*> ] {  
        {[ *SetStateGroup*; ]} {[ *SetStateGroup*; ]} ... {[ *SetStateGroup*; ]}  
    }  
}
```

## Parameters

Item	Description
TechniqueVersion	Either technique10 or technique11. Techniques which use functionality new to Direct3D 11 (5_0 shaders, BindInterfaces, etc) must use technique11.
<i>TechniqueName</i>	Optional. An ASCII string that uniquely identifies the name of the effect technique.
<Annotations>	[in] Optional. One or more pieces of user-supplied information (metadata) that is ignored by the effect system. For syntax, see <a href="#">Annotation Syntax (Direct3D 11)</a> .
pass	Required keyword.
<i>PassName</i>	[in] Optional. An ASCII string that uniquely identifies the name of the pass.
<i>SetStateGroup</i>	[in] Set one or more state groups such as:  <div style="border: 1px solid #ccc; padding: 5px; text-align: center;"><b>StateGroup Syntax</b></div>

Item	Description
	<p><b>StateGroup Syntax</b></p> <p>Blend State</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content; margin: auto;"> <pre>SetBlendState( arguments );</pre> </div> <p>See <a href="#">[ID3D11DeviceContext::OMSetBlendState]</a> (<code>/windows/desktop/api/D3D11/nf-d3d11-id3d11devicecontext-omsetblendstate</code>) for the argument list.</p>
	<p>Depth-stencil State</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content; margin: auto;"> <pre>SetDepthStencilState( arguments );</pre> </div> <p>See <a href="#">ID3D11DeviceContext::OMSetDepthStencilState</a> for the argument list.</p>
	<p>Rasterizer State</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content; margin: auto;"> <pre>SetRasterizerState( arguments );</pre> </div> <p>See <a href="#">[ID3D11DeviceContext::RSSetState]</a> (<code>/windows/desktop/api/D3D11/nf-d3d11-id3d11devicecontext-rssetstate</code>) for the argument list.</p>

Item	Description
	<p><b>StateGroup Syntax</b></p> <p>Shader State</p> <pre data-bbox="679 428 1044 462">SetXXXShader( Shader );</pre> <p>SetXXXShader is one of SetVertexShader, SetDomainShader, SetHullShader, SetGeometryShader, SetPixelShader or SetComputeShader (which are similar to the API methods <a href="#">ID3D11DeviceContext::VSSetShader</a>, <a href="#">ID3D11DeviceContext::DSSetShader</a>, <a href="#">ID3D11DeviceContext::HSSetShader</a>, <a href="#">ID3D11DeviceContext::GSSetShader</a>, <a href="#">ID3D11DeviceContext::PSSetShader</a> and <a href="#">ID3D11DeviceContext::CSSetShader</a>).</p> <p>Shader is a shader variable, which can be obtained in many ways:</p> <pre data-bbox="679 1230 1302 1709"> SetXXXShader( CompileShader(     shader_profile, ShaderFunction( args ) ) ); SetXXXShader( CompileShader( NULL ) ); SetXXXShader( NULL ); SetXXXShader( myShaderVar ); SetXXXShader( myShaderArray[2] ); SetXXXShader( myShaderArray[uIndex] ); SetGeometryShader( ConstructGSWithSO(     Shader, strStream0 ) ); SetGeometryShader( ConstructGSWithSO(     Shader, strStream0, strStream1,     strStream2, strStream3, RastStream ) );</pre>

Item	Description	
	<b>StateGroup Syntax</b>	
Render Target State	One of:	<pre>SetRenderTarget( RTV0, DSV ); SetRenderTarget( RTV0, RTV1, DSV ); ... SetRenderTarget( RTV0, RTV1, RTV2, RTV3, RTV4, RTV5, RTV6, RTV7, DSV );</pre>
		Similar to <a href="#">ID3D11DeviceContext::OMSetRenderTarget</a> .

## Examples

This example sets blending state.

```
BlendState NoBlend
{
    BlendEnable[0] = False;
};

...

technique10
{
    pass p2
    {
        ...
        SetBlendState( NoBlend, float4( 0.0f, 0.0f, 0.0f, 0.0f ), 0xFFFFFFFF
    );
    }
}
```

This example sets up the rasterizer state to render an object in wireframe.

```
RasterizerState rsWireframe { FillMode = WireFrame; };
```

```
...  
  
technique10  
{  
    pass p1  
    {  
        ....  
        SetRasterizerState( rsWireframe );  
    }  
}
```

This example sets shader state.

```
technique10 RenderSceneWithTexture1Light  
{  
    pass P0  
    {  
        SetVertexShader( CompileShader( vs_4_0, RenderSceneVS( 1, true, true  
 ) ) );  
        SetGeometryShader( NULL );  
        SetPixelShader( CompileShader( ps_4_0, RenderScenePS( true ) ) );  
    }  
}
```

## Related topics

[Effect Format](#)

[Effect Group Syntax \(Direct3D 11\)](#)

[Effect State Groups](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Effect State Groups (Direct3D 11)

Article • 05/24/2021

Effect states are name value pairs in the form of an expression.

- Blend State
- Depth and Stencil State
- Rasterizer State
- Sampler State
- Effect Object State
- Defining and using state objects
- Related topics

## Blend State

Effect state	Group
ALPHATOCOVERAGENABLEBLENDENABLESRCBLENDDESTBLENDOP SRCBLENDALPHADESTBLENDALPHABLENDOPALPHARENDERTARGETWRITEMASK	Members of <a href="#">D3D11_BLEND_DESC</a>

## Depth and Stencil State

Effect state
DEPTHENABLEDEPTHWRITEMASKDEPTHFUNCSTENCILENABLESTENCILREADMASKSTENCILWRITEMASK
FRONTFACESTENCILFAILFRONTFACESTENCILZFAILFRONTFACESTENCILPASSFRONTFACESTENCILFUNCBACKFACESTENCILFAILBACKFACESTENCILZFAILBACKFACESTENCI

## Rasterizer State

Effect state	Group
FILLMODE	<a href="#">D3D11_FILL_MODE</a>
CULLMODE	<a href="#">D3D11_CULL_MODE</a>
FRONTCOUNTERCLOCKWISEDEPTHBIASDEPTHBIASCLAMPSLOPESCALEDDEPTHBIAS ZCLIPENABLESCISSORENABLEMULTISAMPLEENABLEANTIALIASINGLINEENABLE	Members of <a href="#">D3D11_RASTERIZER_DESC</a>

## Sampler State

Effect state	Group
Filter AddressU AddressV AddressW MipLODBias MaxAnisotropy ComparisonFunc BorderColor MinLOD MaxLOD	Members of <a href="#">D3D11_SAMPLER_DESC</a>

See [Sampler Type \(DirectX HLSL\)](#) for examples.

## Effect Object State

This Effect Object	Maps to
RASTERIZERSTATE	A <a href="#">Rasterizer State</a> state object.

This Effect Object	Maps to
DEPTHSTENCILSTATE	A Depth and Stencil State state object.
BLENDSTATE	A Blend State state object.
VERTEXSHADER	A compiled vertex shader object.
PIXELSHADER	A compiled pixel shader object.
GEOMETRYSHADER	A compiled geometry shader object.
DS_STENCILREFAB_BLENDFACTORAB_SAMPLEMASK	Members of <a href="#">D3DX11_PASS_DESC</a> .

## Defining and using state objects

State objects are declared in FX files in the following format. StateObjectType is one of the states listed above and MemberName is the name of any member that will have a non-default value.

```
StateObjectType ObjectName {
    MemberName = value;
    ...
    MemberName = value;
};
```

For example, to set up a blend state object with AlphaToCoverageEnable and BlendEnable[0] set to FALSE the following code would be used.

```
BlendState NoBlend {
    AlphaToCoverageEnable = FALSE;
    BlendEnable[0] = FALSE;
};
```

The state object is applied to a technique pass using one of the SetStateGroup functions described in [Effect Technique Syntax \(Direct3D 11\)](#). For example, to apply the BlendState object described above the following code would be used.

```
SetBlendState( NoBlend, float4( 0.0f, 0.0f, 0.0f, 0.0f ), 0xFFFFFFFF );
```

## Related topics

[Effect Technique Syntax](#)

[Effect Format \(Direct3D 11\)](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Effect Group Syntax (Direct3D 11)

Article • 06/20/2024

An effect group is declared with the syntax described in this section.

```
fxgroup GroupName [ <Annotations> ]
{
    TechniqueVersion TechniqueName [ <Annotations> ]
    {
        ...
    }
    TechniqueVersion TechniqueName [ <Annotations> ]
    {
        ...
    }
}
```

## Parameters

[+] Expand table

Item	Description
fxgroup	required keyword.
GroupName	Required. An ASCII string that uniquely identifies the name of the effect group. Unlike techniques, groups must have names to ensure that techniques have a unique identifier (see Groups and Techniques section below).
< Annotations >	[in] Optional. One or more pieces of user-supplied information (metadata) that is ignored by the effect system. For syntax, see Annotation Syntax (Direct3D 11).
TechniqueVersion	Either "technique10" or "technique11". Techniques which use functionality new to Direct3D 11 (5_0 shaders, BindInterfaces, etc) must use "technique11".
TechniqueName	Optional. An ASCII string that uniquely identifies the name of the effect technique.

## Groups and Techniques

In order to maintain compatibility with fx\_4\_0 effects, groups are optional. There is an implicit NULL-named group surrounding all global techniques.

Consider the following example:

```
technique11 GlobalTech
{
}
fxgroup Group1
{
    technique11 Tech1 { ... }
    technique11 Tech2 { ... }
}
fxgroup Group2
{
    technique11 Tech1 { ... }
    technique11 Tech2 { ... }
}
```

In C++, one can get a technique by name in two ways. The following commands will find the obvious techniques:

```
pEffect->GetTechniqueByName( "GlobalTech" );
pEffect->GetTechniqueByName( "|GlobalTech" );
pEffect->GetTechniqueByName( "Group1|Tech1" );
pEffect->GetTechniqueByName( "Group1|Tech2" );
pEffect->GetTechniqueByName( "Group2|Tech1" );
pEffect->GetTechniqueByName( "Group2|Tech2" );
pEffect->GetGroupByName("Group1")->GetTechniqueByName( "Tech1" );
pEffect->GetGroupByName("Group1")->GetTechniqueByName( "Tech2" );
pEffect->GetGroupByName("Group2")->GetTechniqueByName( "Tech1" );
pEffect->GetGroupByName("Group2")->GetTechniqueByName( "Tech2" );
```

In order to ensure that [ID3DX11Effect::GetTechniqueByName](#) works similarly to Effects 10, all defined groups must have a name.

## Related topics

[Effect Format](#)

[Effect Technique Syntax \(Direct3D 11\)](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)