# DXCore

## Adapter Enumeration

DXCore provides a new adapter enumeration API for DirectX devices, superseding DXGI for adapter enumeration purposes. This API is currently experimental, and requires enablement through the Features tab within the [Windows Device Portal](https://docs.microsoft.com/en-us/windows/uwp/debug-test-perf/device-portal) as a prerequisite to be used.

Similar to DXGI, DXCore adapter enumeration begins with creating a factory. However, unlike DXGI, factory creation does not create a snapshot of the adapter state of the system. Rather, DXCore provides a snapshot during adapter list generation time. Should the list become stale due to changing system conditions, the list will become marked as such. You are then able to generate a new, current, adapter list without creating a new factory. Handling these situations is critical to seamlessly respond to events such as adapter arrival and removal, whether it be a GPU, display, or specialized compute adapter, and to appropriately shift workloads in response.

Note that DXCore does not provide any display information. The DisplayMonitor class should be used to retrieve this information where necessary. Adapter LUIDs provide a common identifier to map DXCore adapters to DisplayMonitor.DisplayAdapterId information.

## DXCoreCreateAdapterFactory function

### Syntax

HRESULT  
DXCoreCreateAdapterFactory(  
 \_In\_ REFIID riid,  
 \_COM\_Outptr\_ void \*\*ppFactory  
);

### Parameters

riid

Type: **REFIID**

The global unique identifier (GUID) of the IDXCoreAdapterFactory object referenced by the ppFactory paramter.

ppFactory

Type: **void**\*\*

Address of a pointer to a IDXCoreAdapterFactory object.

### Return Value

Type: **HRESULT**

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT. Will return **E\_NOINTERFACE** if the experimental feature has not been enabled through the Windows Device Portal.

## IDXCoreAdapterFactory interface

The **IDXCoreAdapterFactory** interface implements methods for generating DXCore adapter enumeration objects and retrieving their details.

### Methods

The **IDXCoreAdapterFactory** interface has these methods.

|  |  |
| --- | --- |
| Method | Description |
| IDXCoreAdapterFactory::GetAdapterList | Generate a list representing the current adapter state of the system, filtered by criteria provided. |
| IDXCoreAdapterFactory::GetAdapterByLuid | Retrieves the DXCore adapter object for a specified LUID. |

## IDXCoreAdapterFactory::GetAdapterList method

Generates a list of adapter objects meeting the criteria specified, ordered based on current system conditions and provided preferences.

### Syntax

HRESULT GetAdapterList(  
 \_In\_ const GUID \*filterDXAttributes,   
 \_In\_ uint32\_t numDXAttributes,  
 \_COM\_Outptr\_ IDXCoreAdapterList\*\* adapterList  
);

### Parameters

filterDXAttributes

Type: **const GUID**\*

A pointer to an array of DX attribute GUIDs. At least one GUID must be provided. In the case more than one GUID is provided in the array, only adapters which meet all of the requested attributes will be included in the list.

numDXAttributes

Type: **uint32\_t**

The number of elements in the array pointed at by the filterDXAttributes parameter.

ppAdapterList

Type: **IDXCoreAdapterList**\*\*

Address of a pointer to a IDXCoreAdapterList object.

### Return Value

Type: **HRESULT**

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

### Remarks

Assuming parameters are otherwise valid, this method will return S\_OK even if no adapters are found and create a valid IDXCoreAdapterList objet.

## IDXCoreAdapterFactory::GetAdapterByLuid method

Retrieves the IDXCoreAdapterList object for a specified LUID, if available.

### Syntax

HRESULT GetAdapterByLuid(  
 \_In\_ LUID adapterLUID,  
 \_COM\_Outptr\_ IDXCoreAdapter\*\* adapter  
);

### Parameters

adapterLUID

Type: **LUID**

A unique value that identifies the adapter.

adapter

Type: **IDXCoreAdapter**\*\*

Address of a pointer to a IDXCoreAdapter object.

### Return Value

Type: **HRESULT**

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

## IDXCoreAdapterList interface

The **IDXCoreAdapterList** interface implements methods for retrieving adapter items from a generated list, as well as details about the list.

### Methods

The **IDXCoreAdapterList** interface has these methods.

|  |  |
| --- | --- |
| Method | Description |
| IDXCoreAdapterList::GetItem | Retrieve a specific adapter from the list. |
| IDXCoreAdapterList::GetAdapterCount | Retrieve the number of adapters in the list. |
| IDXCoreAdapterList::IsStale | Evaluate if the generated list is stale due to changed system conditions. |

## IDXCoreAdapterList::GetItem method

Retrieves a specific adapter by index, and optionally its properties, from the IDXCoreAdapterList object.

### Syntax

HRESULT GetItem(  
 \_In\_ uint32\_t index,  
 \_COM\_Outptr\_opt\_ IDXCoreAdapter\*\* adapter  
);

### Parameters

index

Type: **uint32\_t**

The zero based index of the list item you wish to retrieve.

adapter

Type: **IDXCoreAdapter**\*\*

Address of a pointer to a IDXCoreAdapter object.

### Return Value

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

## IDXCoreAdapterList::GetAdapterCount method

Retrieves the number of adapters in the list of the IDXCoreAdapterList object.

### Syntax

uint32\_t GetAdapterCount();

### Parameters

None

### Return Value

Returns the number of items in the list.

## IDXCoreAdapterList::IsStale method

Determine if changes on this system resulted in the adapter list no longer being up to date.

### Syntax

bool IsStale();

### Parameters

None

### Return Value

Returns **true** if system conditions have changed since generating the list that would have impacted the list. Otherwise, returns **false**.

### Remarks

This API can be polled to determine if changing system conditions led to this list no longer being up to date. Once this method returns true, it will continue to do so for the lifetime of the object, even if multiple events have occurred with the end result that the system has later returned to a state identical to when the list was generated.

## IDXCoreAdapter interface

The **IDXCoreAdapter** interface implements methods for retrieving details about an adapter item.

### Methods

The **IDXCoreAdapter** interface has these methods.

|  |  |
| --- | --- |
| Method | Description |
| IDXCoreAdapter::IsValid | Checks if the adapter is still valid. |
| IDXCoreAdapter::IsDXAttributeSupported | Checks if the adapter supports a specified DX attribute. |
| IDXCoreAdapter::GetHardwareID | Retrieves the hardware ID of the adapter. |
| IDXCoreAdapter::GetLUID | Retrieves the LUID of the adapter. |
| IDXCoreAdapter::QueryProperty | Query for an adapter property. |
| IDXCoreAdapter::QueryPropertySize | Determines the size of buffer that is required for a QueryProperty call. |
| IDXCoreAdapter::QueryVideoMemoryInfo | Retrieves information about the processes current budget and usage. |
| IDXCoreAdapter::SetVideoMemoryReservation | Informs the OS of the minimum required physical memory for an application. |

## IDXCoreAdapter::IsValid method

Checks if the adapter is still valid.

### Syntax

bool IsValid();

### Parameters

None

### Return Value

Returns **true** if the adapter is still valid and may be used. Otherwise, returns **false**.

## IDXCoreAdapter::IsDXAttributeSupported method

Checks if the adapter supports a specified DX attribute.

### Syntax

bool IsDXAttributeSupported(  
 \_In\_ GUID attributeGUID  
);

### Parameters

attributeGUID

Type: **GUID**\*

A DX attribute GUID.

### Return Value

Returns **true** if the adapter supports the DX attribute. Otherwise, returns **false**.

## IDXCoreAdapter::GetHardwareID method

Retrieves the hardware ID of the adapter.

### Syntax

HRESULT GetHardwareID(  
 \_Out\_ DXCoreHardwareID \*hardwareID  
);

### Parameters

hardwareID

Type: **DXCoreHardwareID**\*

A pointer to a DXCoreHardwareID struct that will be filled in.

### Return Value

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

## IDXCoreAdapter::GetLUID method

Retrieves the LUID of the adapter.

### Syntax

HRESULT GetLUID(  
 \_Out\_ LUID \*adapterLUID  
);

### Parameters

adapterLUID

Type: **LUID**\*

A pointer to a LUID that will be filled in.

### Return Value

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

## IDXCoreAdapter::QueryProperty method

Query for an adapter property.

### Syntax

HRESULT QueryProperty(  
 \_In\_ DXCoreProperty property,  
 \_In\_ size\_t bufferSize,  
 \_Out\_writes\_(bufferSize) void \*propertyData  
);

### Parameters

property

Type: **DXCoreProperty**

A DXCoreProperty type that you are querying for.

bufferSize

Type: **size\_t**

The size of the buffer you are passing in.

propertyData

Type: **void**\*

A pointer to a caller allocated buffer that will be filled in.

### Return Value

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

## IDXCoreAdapter::QueryPropertySize method

Determines the size of buffer that is required for a QueryProperty call.

### Syntax

HRESULT QueryPropertySize(  
 \_In\_ DXCoreProperty property,  
 \_Out\_ size\_t \*bufferSize  
);

### Parameters

property

Type: **DXCoreProperty**

A DXCoreProperty type that you are querying for.

bufferSize

Type: **size\_t**\*

Pointer to a size\_t variable which will be filled in.

### Return Value

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

## IDXCoreAdapter::QueryVideoMemoryInfo method

Retrieves information about the processes current budget and usage.

### Syntax

HRESULT QueryVideoMemoryInfo(  
 \_In\_ uint32\_t NodeIndex,  
 \_In\_ DXCoreMemorySegmentGroup MemorySegmentGroup,  
 \_Out\_ DXCoreQueryVideoMemoryInfo \*pVideoMemoryInfo  
);

### Parameters

NodeIndex

Type: **uint32\_t**

Specifies the device's physical adapter for which the video memory information is queried. For single-GPU operation, set this to zero. If there are multiple GPU nodes, set this to the index of the node (the device's physical adapter) for which the video memory information is queried. See [Multi-Adapter](https://docs.microsoft.com/en-us/windows/desktop/api/dxgi1_4/nf-dxgi1_4-idxgiadapter3-queryvideomemoryinfo).

MemorySegmentGroup

Type: **DXCoreMemorySegmentGroup**

Specifies a DXCoreMemorySegmentGroup that identifies the group as local or non-local.

pVideoMemoryInfo

Type: **DXCoreQueryVideoMemoryInfo**

Fills in a DXCoreQueryVideoMemoryInfo structure with the current values.

### Return Value

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

### Remarks

This method behaves similarly to the IDXGIAdapter3::QueryVideoMemoryInfo method.

Applications must explicitly manage their usage of physical memory explicitly and keep usage within the budget assigned to the application process. Processes that cannot kept their usage within their assigned budgets will likely experience stuttering, as they are intermittently frozen and paged-out to allow other processes to run.

## IDXCoreAdapter::SetVideoMemoryReservation method

Informs the OS of the minimum required physical memory for an application.

### Syntax

HRESULT SetVideoMemoryReservation(  
 \_In\_ uint32\_t NodeIndex,  
 \_In\_ DXCoreMemorySegmentGroup MemorySegmentGroup,  
 \_In\_ uint64\_t Reservation  
);

### Parameters

NodeIndex

Type: **uint32\_t**

Specifies the device's physical adapter for which the video memory information is queried. For single-GPU operation, set this to zero. If there are multiple GPU nodes, set this to the index of the node (the device's physical adapter) for which the video memory information is queried. See [Multi-Adapter](https://docs.microsoft.com/en-us/windows/desktop/api/dxgi1_4/nf-dxgi1_4-idxgiadapter3-queryvideomemoryinfo).

MemorySegmentGroup

Type: **DXCoreMemorySegmentGroup**

Specifies a DXCoreMemorySegmentGroup that identifies the group as local or non-local.

Reservation

Type: **uint64\_t**

Specifies a value that sets the minimum required physical memory, in bytes.

### Return Value

Returns **S\_OK** if successful; otherwise, returns a failing HRESULT.

### Remarks

This method behaves similarly to the IDXGIAdapter3::QueryVideoMemoryInfo method.

Applications must explicitly manage their usage of physical memory explicitly and keep usage within the budget assigned to the application process. Processes that cannot kept their usage within their assigned budgets will likely experience stuttering, as they are intermittently frozen and paged-out to allow other processes to run.

## DXCoreHardwareID structure

typedef struct DXCoreHardwareID  
{  
 uint32\_t vendorId;  
 uint32\_t deviceId;  
 uint32\_t subSysId;  
 uint32\_t revision;  
} DXCoreHardwareID;

## DXCoreQueryVideoMemoryInfo structure

typedef struct DXCoreQueryVideoMemoryInfo  
{  
 uint64\_t Budget;  
 uint64\_t CurrentUsage;  
 uint64\_t AvailableForReservation;  
 uint64\_t CurrentReservation;  
} DXCoreQueryVideoMemoryInfo;

## DXCoreProperty enum

enum class DXCoreProperty  
{  
 IsHardware = 0,  
 DriverVersion = 1,  
 DriverDescription = 2,  
 KmdModelVersion = 3,  
 IsDriverUpdateInProgress = 4,  
 ComputePreemptionGranularity = 5,  
 GraphicsPreemptionGranularity = 6,  
 DedicatedVideoMemory = 7,  
 ACGCompatible = 8  
};

## DXCoreMemorySegmentGroup enum

enum class DXCoreMemorySegmentGroup  
{  
 Local = 0,  
 NonLocal = 1  
};

## DX Attributes

DXCORE\_ADAPTER\_ATTRIBUTE\_D3D11\_GRFX

This adapter supports being used with the Direct3D 11 graphics APIs. No guarantees are made about specific features, nor is a guarantee made that the OS in its current configuration supports these APIs.

DXCORE\_ADAPTER\_ATTRIBUTE\_D3D12\_GRFX

This adapter supports being used with the Direct3D 12 graphics APIs. No guarantees are made about specific features, nor is a guarantee made that the OS in its current configuration supports these APIs.

DXCORE\_ADAPTER\_ATTRIBUTE\_D3D12\_CORE\_COMPUTE

This adapter supports being used with the Direct3D 12 core compute APIs. No guarantees are made about specific features, nor is a guarantee made that the OS in its current configuration supports these APIs.

### Remarks

A DirectX adapter may support one or more DX attributes. These attributes are used when calling GetAdapterList and IsDXAttributeSupported.

## Sample Code

This sample code shows how to find DirectX adapters that could be used for Direct3D 12 core compute workloads.

HRESULT RunSample()  
{  
 HRESULT hr = S\_OK;  
  
 //  
 // Create a DXCoreAdapterFactory  
 //  
 hr = DXCoreCreateAdapterFactory(\_\_uuidof(IDXCoreAdapterFactory), (void\*\*)&m\_pDXCoreAdapterFactory);  
  
 if (FAILED(hr))  
 {  
 goto cleanup;  
 }  
  
 //  
 // While we wish to keep the app running, attempt to get an  
 // adapter list, and run our workload on an adapter from that list.  
 //  
 while (m\_bKeepAppRunning)  
 {  
 hr = GetAdapterAndRunD3D12CoreCompute();  
 }  
  
cleanup:  
  
 if (m\_pDXCoreAdapterFactory)  
 {  
 m\_pDXCoreAdapterFactory->Release();  
 m\_pDXCoreAdapterFactory = nullptr;  
 }  
  
 return hr;  
}  
  
HRESULT GetAdapterAndRunD3D12CoreCompute()  
{  
 HRESULT hr = S\_OK;  
  
 IDXCoreAdapterList\* pAdapterList = nullptr;  
 IDXCoreAdapter\* pAdapter = nullptr;  
  
 //  
 // Create an adapter list, containing adapters that can be used with D3D12 core compute  
 //  
 const GUID dxGUIDs[] = { DXCORE\_ADAPTER\_ATTRIBUTE\_D3D12\_CORE\_COMPUTE };  
  
 hr = m\_pDXCoreAdapterFactory->GetAdapterList(dxGUIDs,  
 ARRAYSIZE(dxGUIDs),  
 &pAdapterList);  
  
 if (FAILED(hr))  
 {  
 goto cleanup;  
 }  
  
 //  
 // Find the number of adapters in our list, and iterate through them  
 //  
 const uint32\_t ListSize = pAdapterList->GetAdapterCount();  
  
 for (uint32\_t ListIndex = 0; ListIndex < ListSize; ListIndex++)  
 {  
 //  
 // Make sure to release adapters from previous loops  
 //  
 if (pAdapter)  
 {  
 pAdapter->Release();  
 pAdapter = nullptr;  
 }  
  
 //  
 // Retrieve a DXCoreAdapter object for this item in the list  
 //  
 hr = pAdapterList->GetItem(ListIndex, &pAdapter);  
  
 if (FAILED(hr))  
 {  
 continue;  
 }  
  
 //  
 // Check to ensure the adapter hasn't been invalidated since  
 // the list was created.  
 //  
 if (!pAdapter->IsValid())  
 {  
 continue;  
 }  
  
 //  
 // Use our adapter to perform our workload.  
 //  
 hr = DoSampleWorkLoadWithAdapter(pAdapter);  
  
 //  
 // If our workload succeeded, we can exit out.  
 // If it failed, we will try the next adapter.  
 //  
 if (SUCCEEDED(hr))  
 {  
 m\_bKeepAppRunning = false;  
 break;  
 }  
 }  
  
cleanup:  
  
 if (pAdapter)  
 {  
 pAdapter->Release();  
 pAdapter = nullptr;  
 }  
  
 if (pAdapterList)  
 {  
 pAdapterList->Release();  
 pAdapterList = nullptr;  
 }  
  
 return hr;  
}