NAME

Memory Management −

Functions

CUresult cuArray3DCreate (**CUarray** *pHandle, const **CUDA_ARRAY3D_DESCRIPTOR** *pAllocateArray)
Creates a 3D CUDA array.

CUresult cuArray3DGetDescriptor (**CUDA_ARRAY3D_DESCRIPTOR** *pArrayDescriptor, **CUarray** hArray)
Get a 3D CUDA array descriptor.

CUresult cuArrayCreate (**CUarray** *pHandle, const **CUDA_ARRAY_DESCRIPTOR** *pAllocateArray)
Creates a 1D or 2D CUDA array.

CUresult cuArrayDestroy (**CUarray** hArray)
Destroys a CUDA array.

CUresult cuArrayGetDescriptor (**CUDA_ARRAY_DESCRIPTOR** *pArrayDescriptor, **CUarray** hArray)
Get a 1D or 2D CUDA array descriptor.

CUresult cuDeviceGetByPCIBusId (**CUdevice** *dev, const char *pciBusId)
Returns a handle to a compute device.

CUresult cuDeviceGetPCIBusId (char *pciBusId, int len, **CUdevice** dev)
Returns a PCI Bus Id string for the device.

CUresult cuIpcCloseMemHandle (**CUdeviceptr** dptr)
Close memory mapped with **cuIpcOpenMemHandle**.

CUresult cuIpcGetEventHandle (**CUipcEventHandle** *pHandle, **CUevent** event)
Gets an interprocess handle for a previously allocated event.

CUresult cuIpcGetMemHandle (**CUipcMemHandle** *pHandle, **CUdeviceptr** dptr)
Gets an interprocess memory handle for an existing device memory allocation.

CUresult cuIpcOpenEventHandle (**CUevent** *phEvent, **CUipcEventHandle** handle)
Opens an interprocess event handle for use in the current process.

CUresult cuIpcOpenMemHandle (**CUdeviceptr** *pdptr, **CUipcMemHandle** handle, unsigned int Flags)
Opens an interprocess memory handle exported from another process and returns a device pointer usable in the local process.

CUresult cuMemAlloc (**CUdeviceptr** *dptr, size_t bytesize)
Allocates device memory.

CUresult cuMemAllocHost (void **pp, size_t bytesize)
Allocates page-locked host memory.

CUresult cuMemAllocManaged (**CUdeviceptr** *dptr, size_t bytesize, unsigned int flags)
Allocates memory that will be automatically managed by the Unified Memory system.

CUresult cuMemAllocPitch (**CUdeviceptr** *dptr, size_t *pPitch, size_t WidthInBytes, size_t Height, unsigned int ElementSizeBytes)
Allocates pitched device memory.

CUresult cuMemcpy (**CUdeviceptr** dst, **CUdeviceptr** src, size_t ByteCount)
Copies memory.

CUresult cuMemcpy2D (const **CUDA_MEMCPY2D** *pCopy)
Copies memory for 2D arrays.

CUresult cuMemcpy2DAsync (const **CUDA_MEMCPY2D** *pCopy, **CUstream** hStream)
Copies memory for 2D arrays.

CUresult cuMemcpy2DUnaligned (const **CUDA_MEMCPY2D** *pCopy)
Copies memory for 2D arrays.

CUresult cuMemcpy3D (const **CUDA_MEMCPY3D** *pCopy)
Copies memory for 3D arrays.

CUresult cuMemcpy3DAsync (const **CUDA_MEMCPY3D** *pCopy, **CUstream** hStream)
Copies memory for 3D arrays.

CUresult cuMemcpy3DPeer (const **CUDA_MEMCPY3D_PEER** *pCopy)
Copies memory between contexts.

CUresult cuMemcpy3DPeerAsync (const **CUDA_MEMCPY3D_PEER** *pCopy, **CUstream** hStream)



Copies memory between contexts asynchronously.

CUresult cuMemcpyAsync (**CUdeviceptr** dst, **CUdeviceptr** src, size_t ByteCount, **CUstream** hStream)
Copies memory asynchronously.

CUresult cuMemcpyAtoA (**CUarray** dstArray, size_t dstOffset, **CUarray** srcArray, size_t srcOffset, size_t ByteCount)
Copies memory from Array to Array.

CUresult cuMemcpyAtoD (**CUdeviceptr** dstDevice, **CUarray** srcArray, size_t srcOffset, size_t ByteCount)
Copies memory from Array to Device.

CUresult cuMemcpyAtoH (void *dstHost, **CUarray** srcArray, size_t srcOffset, size_t ByteCount)
Copies memory from Array to Host.

CUresult cuMemcpyAtoHAsync (void *dstHost, **CUarray** srcArray, size_t srcOffset, size_t ByteCount, **CUstream** hStream)
Copies memory from Array to Host.

CUresult cuMemcpyDtoA (**CUarray** dstArray, size_t dstOffset, **CUdeviceptr** srcDevice, size_t ByteCount)
Copies memory from Device to Array.

CUresult cuMemcpyDtoD (**CUdeviceptr** dstDevice, **CUdeviceptr** srcDevice, size_t ByteCount)
Copies memory from Device to Device.

CUresult cuMemcpyDtoDAsync (**CUdeviceptr** dstDevice, **CUdeviceptr** srcDevice, size_t ByteCount, **CUstream** hStream)
Copies memory from Device to Device.

CUresult cuMemcpyDtoH (void *dstHost, **CUdeviceptr** srcDevice, size_t ByteCount)
Copies memory from Device to Host.

CUresult cuMemcpyDtoHAsync (void *dstHost, **CUdeviceptr** srcDevice, size_t ByteCount, **CUstream** hStream)
Copies memory from Device to Host.

CUresult cuMemcpyHtoA (**CUarray** dstArray, size_t dstOffset, const void *srcHost, size_t ByteCount)
Copies memory from Host to Array.

CUresult cuMemcpyHtoAAsync (**CUarray** dstArray, size_t dstOffset, const void *srcHost, size_t ByteCount, **CUstream** hStream)
Copies memory from Host to Array.

CUresult cuMemcpyHtoD (**CUdeviceptr** dstDevice, const void *srcHost, size_t ByteCount)
Copies memory from Host to Device.

CUresult cuMemcpyHtoDAsync (**CUdeviceptr** dstDevice, const void *srcHost, size_t ByteCount, **CUstream** hStream)
Copies memory from Host to Device.

CUresult cuMemcpyPeer (**CUdeviceptr** dstDevice, **CUcontext** dstContext, **CUdeviceptr** srcDevice, **CUcontext** srcContext, size_t ByteCount)
Copies device memory between two contexts.

CUresult cuMemcpyPeerAsync (**CUdeviceptr** dstDevice, **CUcontext** dstContext, **CUdeviceptr** srcDevice, **CUcontext** srcContext, size_t ByteCount, **CUstream** hStream)
Copies device memory between two contexts asynchronously.

CUresult cuMemFree (**CUdeviceptr** dptr)
Frees device memory.

CUresult cuMemFreeHost (void *p)
Frees page-locked host memory.

CUresult cuMemGetAddressRange (**CUdeviceptr** *pbase, size_t *psize, **CUdeviceptr** dptr)
Get information on memory allocations.

CUresult cuMemGetInfo (size_t *free, size_t *total)
Gets free and total memory.

CUresult cuMemHostAlloc (void **pp, size_t bytesize, unsigned int Flags)
Allocates page-locked host memory.

CUresult cuMemHostGetDevicePointer (**CUdeviceptr** *pdptr, void *p, unsigned int Flags)
Passes back device pointer of mapped pinned memory.

CUresult cuMemHostGetFlags (unsigned int *pFlags, void *p)
Passes back flags that were used for a pinned allocation.



CUresult cuMemHostRegister (void *p, size_t bytesize, unsigned int Flags)
Registers an existing host memory range for use by CUDA.

CUresult cuMemHostUnregister (void *p)
Unregisters a memory range that was registered with cuMemHostRegister.

CUresult cuMemsetD16 (**CUdeviceptr** dstDevice, unsigned short us, size_t N)
Initializes device memory.

CUresult cuMemsetD16Async (**CUdeviceptr** dstDevice, unsigned short us, size_t N, **CUstream** hStream)
Sets device memory.

CUresult cuMemsetD2D16 (**CUdeviceptr** dstDevice, size_t dstPitch, unsigned short us, size_t Width, size_t Height)
Initializes device memory.

CUresult cuMemsetD2D16Async (**CUdeviceptr** dstDevice, size_t dstPitch, unsigned short us, size_t Width, size_t Height, **CUstream** hStream)
Sets device memory.

CUresult cuMemsetD2D32 (**CUdeviceptr** dstDevice, size_t dstPitch, unsigned int ui, size_t Width, size_t Height)
Initializes device memory.

CUresult cuMemsetD2D32Async (**CUdeviceptr** dstDevice, size_t dstPitch, unsigned int ui, size_t Width, size_t Height, **CUstream** hStream)
Sets device memory.

CUresult cuMemsetD2D8 (**CUdeviceptr** dstDevice, size_t dstPitch, unsigned char uc, size_t Width, size_t Height)
Initializes device memory.

CUresult cuMemsetD2D8Async (**CUdeviceptr** dstDevice, size_t dstPitch, unsigned char uc, size_t Width, size_t Height, **CUstream** hStream)
Sets device memory.

CUresult cuMemsetD32 (**CUdeviceptr** dstDevice, unsigned int ui, size_t N)
Initializes device memory.

CUresult cuMemsetD32Async (**CUdeviceptr** dstDevice, unsigned int ui, size_t N, **CUstream** hStream)
Sets device memory.

CUresult cuMemsetD8 (**CUdeviceptr** dstDevice, unsigned char uc, size_t N)
Initializes device memory.

CUresult cuMemsetD8Async (**CUdeviceptr** dstDevice, unsigned char uc, size_t N, **CUstream** hStream)
Sets device memory.

CUresult cuMipmappedArrayCreate (**CUmipmappedArray** *pHandle, const **CUDA_ARRAY3D_DESCRIPTOR** *pMipmappedArrayDesc, unsigned int numMipmapLevels)
Creates a CUDA mipmapped array.

CUresult cuMipmappedArrayDestroy (**CUmipmappedArray** hMipmappedArray)
Destroys a CUDA mipmapped array.

CUresult cuMipmappedArrayGetLevel (**CUarray** *pLevelArray, **CUmipmappedArray** hMipmappedArray, unsigned int level)
Gets a mipmap level of a CUDA mipmapped array.

Detailed Description

\brief memory management functions of the low-level CUDA driver API (**cuda.h**)

This section describes the memory management functions of the low-level CUDA driver application programming interface.

Function Documentation

CUresult cuArray3DCreate (CUarray * pHandle, const CUDA_ARRAY3D_DESCRIPTOR * pAllocateArray)

Creates a CUDA array according to the **CUDA_ARRAY3D_DESCRIPTOR** structure `pAllocateArray` and returns a handle to the new CUDA array in `*pHandle`. The **CUDA_ARRAY3D_DESCRIPTOR** is defined as:

```
typedef struct {
```



```
    unsigned int Width;
    unsigned int Height;
    unsigned int Depth;
    CUarray_format Format;
    unsigned int NumChannels;
    unsigned int Flags;
} CUDA_ARRAY3D_DESCRIPTOR;
```

where:

- `Width`, `Height`, and `Depth` are the width, height, and depth of the CUDA array (in elements); the following types of CUDA arrays can be allocated:
  - A 1D array is allocated if `Height` and `Depth` extents are both zero.
  - A 2D array is allocated if only `Depth` extent is zero.
  - A 3D array is allocated if all three extents are non-zero.
  - A 1D layered CUDA array is allocated if only `Height` is zero and the **CUDA_ARRAY3D_LAYERED** flag is set. Each layer is a 1D array. The number of layers is determined by the depth extent.
  - A 2D layered CUDA array is allocated if all three extents are non-zero and the **CUDA_ARRAY3D_LAYERED** flag is set. Each layer is a 2D array. The number of layers is determined by the depth extent.
  - A cubemap CUDA array is allocated if all three extents are non-zero and the **CUDA_ARRAY3D_CUBEMAP** flag is set. `Width` must be equal to `Height`, and `Depth` must be six. A cubemap is a special type of 2D layered CUDA array, where the six layers represent the six faces of a cube. The order of the six layers in memory is the same as that listed in **CUarray_cubemap_face**.
  - A cubemap layered CUDA array is allocated if all three extents are non-zero, and both, **CUDA_ARRAY3D_CUBEMAP** and **CUDA_ARRAY3D_LAYERED** flags are set. `Width` must be equal to `Height`, and `Depth` must be a multiple of six. A cubemap layered CUDA array is a special type of 2D layered CUDA array that consists of a collection of cubemaps. The first six layers represent the first cubemap, the next six layers form the second cubemap, and so on.
- Format specifies the format of the elements; **CUarray_format** is defined as:

```
typedef enum CUarray_format_enum {
    CU_AD_FORMAT_UNSIGNED_INT8 = 0x01,
    CU_AD_FORMAT_UNSIGNED_INT16 = 0x02,
    CU_AD_FORMAT_UNSIGNED_INT32 = 0x03,
    CU_AD_FORMAT_SIGNED_INT8 = 0x08,
    CU_AD_FORMAT_SIGNED_INT16 = 0x09,
    CU_AD_FORMAT_SIGNED_INT32 = 0x0a,
    CU_AD_FORMAT_HALF = 0x10,
    CU_AD_FORMAT_FLOAT = 0x20
} CUarray_format;
```

- `NumChannels` specifies the number of packed components per CUDA array element; it may be 1, 2, or 4;
- Flags may be set to
  - **CUDA_ARRAY3D_LAYERED** to enable creation of layered CUDA arrays. If this flag is set, `Depth` specifies the number of layers, not the depth of a 3D array.
  - **CUDA_ARRAY3D_SURFACE_LDST** to enable surface references to be bound to the CUDA array. If this flag is not set, **cuSurfRefSetArray** will fail when attempting to bind the CUDA array to a surface reference.
  - **CUDA_ARRAY3D_CUBEMAP** to enable creation of cubemaps. If this flag is set, `Width` must be equal to `Height`, and `Depth` must be six. If the **CUDA_ARRAY3D_LAYERED** flag is also set, then `Depth` must be a multiple of six.



- **CUDA_ARRAY3D_TEXTURE_GATHER** to indicate that the CUDA array will be used for texture gather. Texture gather can only be performed on 2D CUDA arrays.

`Width`, `Height` and `Depth` must meet certain size requirements as listed in the following table. All values are specified in elements. Note that for brevity's sake, the full name of the device attribute is not specified. For ex., TEXTURE1D_WIDTH refers to the device attribute **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_WIDTH**.

Note that 2D CUDA arrays have different size requirements if the **CUDA_ARRAY3D_TEXTURE_GATHER** flag is set. `Width` and `Height` must not be greater than **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_GATHER_WIDTH** and **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_GATHER_HEIGHT** respectively, in that case.

CUDA array type Valid extents that must always be met {(width range in elements), (height range), (depth range)} Valid extents with CUDA_ARRAY3D_SURFACE_LDST set {(width range in elements), (height range), (depth range)} 1D { (1,TEXTURE1D_WIDTH), 0, 0 } { (1,SURFACE1D_WIDTH), 0, 0 } 2D { (1,TEXTURE2D_WIDTH), (1,TEXTURE2D_HEIGHT), 0 } { (1,SURFACE2D_WIDTH), (1,SURFACE2D_HEIGHT), 0 } 3D { (1,TEXTURE3D_WIDTH), (1,TEXTURE3D_HEIGHT), (1,TEXTURE3D_DEPTH) }
OR
{ (1,TEXTURE3D_WIDTH_ALTERNATE), (1,TEXTURE3D_HEIGHT_ALTERNATE), (1,TEXTURE3D_DEPTH_ALTERNATE) } { (1,SURFACE3D_WIDTH), (1,SURFACE3D_HEIGHT), (1,SURFACE3D_DEPTH) } 1D Layered { (1,TEXTURE1D_LAYERED_WIDTH), 0, (1,TEXTURE1D_LAYERED_LAYERS) } { (1,SURFACE1D_LAYERED_WIDTH), 0, (1,SURFACE1D_LAYERED_LAYERS) } 2D Layered { (1,TEXTURE2D_LAYERED_WIDTH), (1,TEXTURE2D_LAYERED_HEIGHT), (1,TEXTURE2D_LAYERED_LAYERS) } { (1,SURFACE2D_LAYERED_WIDTH), (1,SURFACE2D_LAYERED_HEIGHT), (1,SURFACE2D_LAYERED_LAYERS) } Cubemap { (1,TEXTURECUBEMAP_WIDTH), (1,TEXTURECUBEMAP_WIDTH), 6 } { (1,SURFACECUBEMAP_WIDTH), (1,SURFACECUBEMAP_WIDTH), 6 } Cubemap Layered { (1,TEXTURECUBEMAP_LAYERED_WIDTH), (1,TEXTURECUBEMAP_LAYERED_WIDTH), (1,TEXTURECUBEMAP_LAYERED_LAYERS) } { (1,SURFACECUBEMAP_LAYERED_WIDTH), (1,SURFACECUBEMAP_LAYERED_WIDTH), (1,SURFACECUBEMAP_LAYERED_LAYERS) }

Here are examples of CUDA array descriptions:

Description for a CUDA array of 2048 floats:

```
CUDA_ARRAY3D_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 2048;
desc.Height = 0;
desc.Depth = 0;
```

Description for a 64 x 64 CUDA array of floats:

```
CUDA_ARRAY3D_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 64;
desc.Height = 64;
desc.Depth = 0;
```

Description for a `width` x `height` x `depth` CUDA array of 64-bit, 4x16-bit float16's:

```
CUDA_ARRAY3D_DESCRIPTOR desc;
desc.FormatFlags = CU_AD_FORMAT_HALF;
desc.NumChannels = 4;
desc.Width = width;
```



```
desc.Height = height;
desc.Depth = depth;
```

Parameters:

pHandle - Returned array
pAllocateArray - 3D array descriptor

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**, **CUDA_ERROR_UNKNOWN**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DGetDescriptor, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuArray3DGetDescriptor (CUDA_ARRAY3D_DESCRIPTOR * pArrayDescriptor, CUarray hArray)

Returns in `*pArrayDescriptor` a descriptor containing information on the format and dimensions of the CUDA array `hArray`. It is useful for subroutines that have been passed a CUDA array, but need to know the CUDA array parameters for validation or other purposes.

This function may be called on 1D and 2D arrays, in which case the `Height` and/or `Depth` members of the descriptor struct will be set to 0.

Parameters:

pArrayDescriptor - Returned 3D array descriptor
hArray - 3D array to get descriptor of

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_INVALID_HANDLE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuArrayCreate (CUarray * pHandle, const CUDA_ARRAY_DESCRIPTOR * pAllocateArray)

Creates a CUDA array according to the **CUDA_ARRAY_DESCRIPTOR** structure `pAllocateArray` and returns a handle to the new CUDA array in `*pHandle`. The **CUDA_ARRAY_DESCRIPTOR** is defined as:



```
typedef struct {
    unsigned int Width;
    unsigned int Height;
    CUarray_format Format;
    unsigned int NumChannels;
} CUDA_ARRAY_DESCRIPTOR;
```

where:

- `Width`, and `Height` are the width, and height of the CUDA array (in elements); the CUDA array is one-dimensional if height is 0, two-dimensional otherwise;
- Format specifies the format of the elements; **CUarray_format** is defined as:

```
typedef enum CUarray_format_enum {
    CU_AD_FORMAT_UNSIGNED_INT8 = 0x01,
    CU_AD_FORMAT_UNSIGNED_INT16 = 0x02,
    CU_AD_FORMAT_UNSIGNED_INT32 = 0x03,
    CU_AD_FORMAT_SIGNED_INT8 = 0x08,
    CU_AD_FORMAT_SIGNED_INT16 = 0x09,
    CU_AD_FORMAT_SIGNED_INT32 = 0x0a,
    CU_AD_FORMAT_HALF = 0x10,
    CU_AD_FORMAT_FLOAT = 0x20
} CUarray_format;
```

- `NumChannels` specifies the number of packed components per CUDA array element; it may be 1, 2, or 4;

Here are examples of CUDA array descriptions:

Description for a CUDA array of 2048 floats:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 2048;
desc.Height = 1;
```

Description for a 64 x 64 CUDA array of floats:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 64;
desc.Height = 64;
```

Description for a `width` x `height` CUDA array of 64-bit, 4x16-bit float16's:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.FormatFlags = CU_AD_FORMAT_HALF;
desc.NumChannels = 4;
desc.Width = width;
desc.Height = height;
```

Description for a `width` x `height` CUDA array of 16-bit elements, each of which is two 8-bit unsigned chars:

```
CUDA_ARRAY_DESCRIPTOR arrayDesc;
desc.FormatFlags = CU_AD_FORMAT_UNSIGNED_INT8;
desc.NumChannels = 2;
desc.Width = width;
desc.Height = height;
```



Parameters:
pHandle - Returned array
pAllocateArray - Array descriptor

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**, **CUDA_ERROR_UNKNOWN**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuArrayDestroy (CUarray hArray)
Destroys the CUDA array `hArray`.

Parameters:
hArray - Array to destroy

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_HANDLE**, **CUDA_ERROR_ARRAY_IS_MAPPED**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuArrayGetDescriptor (CUDA_ARRAY_DESCRIPTOR * pArrayDescriptor, CUarray hArray)
Returns in `*pArrayDescriptor` a descriptor containing information on the format and dimensions of the CUDA array `hArray`. It is useful for subroutines that have been passed a CUDA array, but need to know the CUDA array parameters for validation or other purposes.

Parameters:
pArrayDescriptor - Returned array descriptor
hArray - Array to get descriptor of

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_INVALID_HANDLE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.



See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuDeviceGetByPCIBusId (CUdevice * dev, const char * pciBusId)

Returns in `*device` a device handle given a PCI bus ID string.

Parameters:

dev - Returned device handle
pciBusId - String in one of the following forms: [domain]:[bus]:[device].[function] [domain]:[bus]:[device] [bus]:[device].[function] where `domain`, `bus`, `device`, and `function` are all hexadecimal values

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_INVALID_DEVICE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceGet, **cuDeviceGetAttribute**, **cuDeviceGetPCIBusId**

CUresult cuDeviceGetPCIBusId (char * pciBusId, int len, CUdevice dev)

Returns an ASCII string identifying the device `dev` in the NULL-terminated string pointed to by `pciBusId`. `len` specifies the maximum length of the string that may be returned.

Parameters:

pciBusId - Returned identifier string for the device in the following format [domain]:[bus]:[device].[function] where `domain`, `bus`, `device`, and `function` are all hexadecimal values. pciBusId should be large enough to store 13 characters including the NULL-terminator.
len - Maximum length of string to store in `name`
dev - Device to get identifier string for

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_INVALID_DEVICE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuDeviceGet, **cuDeviceGetAttribute**, **cuDeviceGetByPCIBusId**

CUresult cuIpcCloseMemHandle (CUdeviceptr dptr)

Unmaps memory returnd by **cuIpcOpenMemHandle**. The original allocation in the exporting process as well as imported mappings in other processes will be unaffected.

Any resources used to enable peer access will be freed if this is the last mapping using them.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:

dptr - Device pointer returned by **cuIpcOpenMemHandle**

Returns:

CUDA_SUCCESS, **CUDA_ERROR_INVALID_CONTEXT**,



CUDA_ERROR_MAP_FAILED, **CUDA_ERROR_INVALID_HANDLE**,

See also:

cuMemAlloc, **cuMemFree**, **cuIpcGetEventHandle**, **cuIpcOpenEventHandle**, **cuIpcGetMemHandle**, **cuIpcOpenMemHandle**,

CUresult cuIpcGetEventHandle (CUipcEventHandle * pHandle, CUevent event)

Takes as input a previously allocated event. This event must have been created with the **CU_EVENT_INTERPROCESS** and **CU_EVENT_DISABLE_TIMING** flags set. This opaque handle may be copied into other processes and opened with **cuIpcOpenEventHandle** to allow efficient hardware synchronization between GPU work in different processes.

After the event has been opened in the importing process, **cuEventRecord**, **cuEventSynchronize**, **cuStreamWaitEvent** and **cuEventQuery** may be used in either process. Performing operations on the imported event after the exported event has been freed with **cuEventDestroy** will result in undefined behavior.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:

pHandle - Pointer to a user allocated **CUipcEventHandle** in which to return the opaque event handle
event - Event allocated with **CU_EVENT_INTERPROCESS** and **CU_EVENT_DISABLE_TIMING** flags.

Returns:

CUDA_SUCCESS, **CUDA_ERROR_INVALID_HANDLE**, **CUDA_ERROR_OUT_OF_MEMORY**, **CUDA_ERROR_MAP_FAILED**

See also:

cuEventCreate, **cuEventDestroy**, **cuEventSynchronize**, **cuEventQuery**, **cuStreamWaitEvent**, **cuIpcOpenEventHandle**, **cuIpcGetMemHandle**, **cuIpcOpenMemHandle**, **cuIpcCloseMemHandle**

CUresult cuIpcGetMemHandle (CUipcMemHandle * pHandle, CUdeviceptr dptr)

Takes a pointer to the base of an existing device memory allocation created with **cuMemAlloc** and exports it for use in another process. This is a lightweight operation and may be called multiple times on an allocation without adverse effects.

If a region of memory is freed with **cuMemFree** and a subsequent call to **cuMemAlloc** returns memory with the same device address, **cuIpcGetMemHandle** will return a unique handle for the new memory.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:

pHandle - Pointer to user allocated **CUipcMemHandle** to return the handle in.
dptr - Base pointer to previously allocated device memory

Returns:

CUDA_SUCCESS, **CUDA_ERROR_INVALID_HANDLE**, **CUDA_ERROR_OUT_OF_MEMORY**, **CUDA_ERROR_MAP_FAILED**,

See also:

cuMemAlloc, **cuMemFree**, **cuIpcGetEventHandle**, **cuIpcOpenEventHandle**, **cuIpcOpenMemHandle**, **cuIpcCloseMemHandle**

CUresult cuIpcOpenEventHandle (CUevent * phEvent, CUipcEventHandle handle)

Opens an interprocess event handle exported from another process with **cuIpcGetEventHandle**. This function returns a **CUevent** that behaves like a locally created event with the **CU_EVENT_DISABLE_TIMING** flag specified. This event must be freed with **cuEventDestroy**.

Performing operations on the imported event after the exported event has been freed with **cuEventDestroy** will result in undefined behavior.

IPC functionality is restricted to devices with support for unified addressing on Linux operating



systems.

Parameters:

phEvent - Returns the imported event
handle - Interprocess handle to open

Returns:

CUDA_SUCCESS, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_MAP_FAILED**, **CUDA_ERROR_PEER_ACCESS_UNSUPPORTED**, **CUDA_ERROR_INVALID_HANDLE**

See also:

cuEventCreate, **cuEventDestroy**, **cuEventSynchronize**, **cuEventQuery**, **cuStreamWaitEvent**, **cuIpcGetEventHandle**, **cuIpcGetMemHandle**, **cuIpcOpenMemHandle**, **cuIpcCloseMemHandle**

CUresult cuIpcOpenMemHandle (CUdeviceptr * pdptr, CUipcMemHandle handle, unsigned int Flags)

Maps memory exported from another process with **cuIpcGetMemHandle** into the current device address space. For contexts on different devices **cuIpcOpenMemHandle** can attempt to enable peer access between the devices as if the user called **cuCtxEnablePeerAccess**. This behavior is controlled by the **CU_IPC_MEM_LAZY_ENABLE_PEER_ACCESS** flag. **cuDeviceCanAccessPeer** can determine if a mapping is possible.

Contexts that may open CUipcMemHandles are restricted in the following way. CUipcMemHandles from each **CUdevice** in a given process may only be opened by one **CUcontext** per **CUdevice** per other process.

Memory returned from **cuIpcOpenMemHandle** must be freed with **cuIpcCloseMemHandle**.

Calling **cuMemFree** on an exported memory region before calling **cuIpcCloseMemHandle** in the importing context will result in undefined behavior.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:

pdptr - Returned device pointer
handle - **CUipcMemHandle** to open
Flags - Flags for this operation. Must be specified as **CU_IPC_MEM_LAZY_ENABLE_PEER_ACCESS**

Returns:

CUDA_SUCCESS, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_MAP_FAILED**, **CUDA_ERROR_INVALID_HANDLE**, **CUDA_ERROR_TOO_MANY_PEERS**

Note:

No guarantees are made about the address returned in `*pdptr`. In particular, multiple processes may not receive the same address for the same `handle`.

See also:

cuMemAlloc, **cuMemFree**, **cuIpcGetEventHandle**, **cuIpcOpenEventHandle**, **cuIpcGetMemHandle**, **cuIpcCloseMemHandle**, **cuCtxEnablePeerAccess**, **cuDeviceCanAccessPeer**,

CUresult cuMemAlloc (CUdeviceptr * dptr, size_t bytesize)

Allocates `bytesize` bytes of linear memory on the device and returns in `*dptr` a pointer to the allocated memory. The allocated memory is suitably aligned for any kind of variable. The memory is not cleared. If `bytesize` is 0, **cuMemAlloc()** returns **CUDA_ERROR_INVALID_VALUE**.

Parameters:

dptr - Returned device pointer
bytesize - Requested allocation size in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**,



CUDA_ERROR_INVALID_VALUE, **CUDA_ERROR_OUT_OF_MEMORY**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemAllocHost (void ** pp, size_t bytesize)

Allocates `bytesize` bytes of host memory that is page-locked and accessible to the device. The driver tracks the virtual memory ranges allocated with this function and automatically accelerates calls to functions such as **cuMemcpy()**. Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as malloc(). Allocating excessive amounts of memory with **cuMemAllocHost()** may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to allocate staging areas for data exchange between host and device.

Note all host memory allocated using **cuMemHostAlloc()** will automatically be immediately accessible to all contexts on all devices which support unified addressing (as may be queried using **CU_DEVICE_ATTRIBUTE_UNIFIED_ADDRESSING**). The device pointer that may be used to access this host memory from those contexts is always equal to the returned host pointer `*pp`. See **Unified Addressing** for additional details.

Parameters:

pp - Returned host pointer to page-locked memory
bytesize - Requested allocation size in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemAllocManaged (CUdeviceptr * dptr, size_t bytesize, unsigned int flags)

Allocates `bytesize` bytes of managed memory on the device and returns in `*dptr` a pointer to the allocated memory. If the device doesn't support allocating managed memory, **CUDA_ERROR_NOT_SUPPORTED** is returned. Support for managed memory can be queried using the device attribute **CU_DEVICE_ATTRIBUTE_MANAGED_MEMORY**. The allocated memory is suitably aligned for any kind of variable. The memory is not cleared. If `bytesize` is 0, **cuMemAllocManaged** returns **CUDA_ERROR_INVALID_VALUE**. The pointer is valid on the CPU and on all GPUs in the system that support managed memory. All accesses to this pointer must obey the Unified Memory programming model.

`flags` specifies the default stream association for this allocation. `flags` must be one of



CU_MEM_ATTACH_GLOBAL or **CU_MEM_ATTACH_HOST**. If **CU_MEM_ATTACH_GLOBAL** is specified, then this memory is accessible from any stream on any device. If **CU_MEM_ATTACH_HOST** is specified, then the allocation is created with initial visibility restricted to host access only; an explicit call to **cuStreamAttachMemAsync** will be required to enable access on the device.

If the association is later changed via **cuStreamAttachMemAsync** to a single stream, the default association as specified during **cuMemAllocManaged** is restored when that stream is destroyed. For __managed__ variables, the default association is always **CU_MEM_ATTACH_GLOBAL**. Note that destroying a stream is an asynchronous operation, and as a result, the change to default association won't happen until all work in the stream has completed.

Memory allocated with **cuMemAllocManaged** should be released with **cuMemFree**.

On a multi-GPU system with peer-to-peer support, where multiple GPUs support managed memory, the physical storage is created on the GPU which is active at the time **cuMemAllocManaged** is called. All other GPUs will reference the data at reduced bandwidth via peer mappings over the PCIe bus. The Unified Memory management system does not migrate memory between GPUs.

On a multi-GPU system where multiple GPUs support managed memory, but not all pairs of such GPUs have peer-to-peer support between them, the physical storage is created in 'zero-copy' or system memory. All GPUs will reference the data at reduced bandwidth over the PCIe bus. In these circumstances, use of the environment variable, CUDA_VISIBLE_DEVICES, is recommended to restrict CUDA to only use those GPUs that have peer-to-peer support. Alternatively, users can also set CUDA_MANAGED_FORCE_DEVICE_ALLOC to a non-zero value to force the driver to always use device memory for physical storage. When this environment variable is set to a non-zero value, all contexts created in that process on devices that support managed memory have to be peer-to-peer compatible with each other. Context creation will fail if a context is created on a device that supports managed memory and is not peer-to-peer compatible with any of the other managed memory supporting devices on which contexts were previously created, even if those contexts have been destroyed. These environment variables are described in the CUDA programming guide under the 'CUDA environment variables' section.

Parameters:

dptr - Returned device pointer
bytesize - Requested allocation size in bytes
flags - Must be one of **CU_MEM_ATTACH_GLOBAL** or **CU_MEM_ATTACH_HOST**

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_NOT_SUPPORTED**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**, **cuDeviceGetAttribute**, **cuStreamAttachMemAsync**

CUresult cuMemAllocPitch (CUdeviceptr * dptr, size_t * pPitch, size_t WidthInBytes, size_t Height, unsigned int ElementSizeBytes)

Allocates at least `WidthInBytes * Height` bytes of linear memory on the device and returns in `*dptr` a pointer to the allocated memory. The function may pad the allocation to ensure that corresponding pointers in any given row will continue to meet the alignment requirements for



coalescing as the address is updated from row to row. `ElementSizeBytes` specifies the size of the largest reads and writes that will be performed on the memory range. `ElementSizeBytes` may be 4, 8 or 16 (since coalesced memory transactions are not possible on other data sizes). If `ElementSizeBytes` is smaller than the actual read/write size of a kernel, the kernel will run correctly, but possibly at reduced speed. The pitch returned in `*pPitch` by **cuMemAllocPitch()** is the width in bytes of the allocation. The intended usage of pitch is as a separate parameter of the allocation, used to compute addresses within the 2D array. Given the row and column of an array element of type **T**, the address is computed as:

```
T* pElement = (T*)((char*)BaseAddress + Row * Pitch) + Column;
```

The pitch returned by **cuMemAllocPitch()** is guaranteed to work with **cuMemcpy2D()** under all circumstances. For allocations of 2D arrays, it is recommended that programmers consider performing pitch allocations using **cuMemAllocPitch()**. Due to alignment restrictions in the hardware, this is especially true if the application will be performing 2D memory copies between different regions of device memory (whether linear memory or CUDA arrays).

The byte alignment of the pitch returned by **cuMemAllocPitch()** is guaranteed to match or exceed the alignment requirement for texture binding with **cuTexRefSetAddress2D()**.

Parameters:

dptr - Returned device pointer
pPitch - Returned pitch of allocation in bytes
WidthInBytes - Requested allocation width in bytes
Height - Requested allocation height in rows
ElementSizeBytes - Size of largest reads/writes for range

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpy (CUdeviceptr dst, CUdeviceptr src, size_t ByteCount)

Copies data between two pointers. `dst` and `src` are base pointers of the destination and source, respectively. `ByteCount` specifies the number of bytes to copy. Note that this function infers the type of the transfer (host to host, host to device, device to device, or device to host) from the pointer values. This function is only allowed in contexts which support unified addressing.

Parameters:

dst - Destination unified virtual address space pointer
src - Source unified virtual address space pointer
ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.



This function exhibits behavior for most use cases.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpy2D (const CUDA_MEMCPY2D * pCopy)

Perform a 2D memory copy according to the parameters specified in `pCopy`. The **CUDA_MEMCPY2D** structure is defined as:

```
typedef struct CUDA_MEMCPY2D_st {
  unsigned int srcXInBytes, srcY;
  CUmemorytype srcMemoryType;
    const void *srcHost;
    CUdeviceptr srcDevice;
    CUarray srcArray;
    unsigned int srcPitch;

  unsigned int dstXInBytes, dstY;
  CUmemorytype dstMemoryType;
    void *dstHost;
    CUdeviceptr dstDevice;
    CUarray dstArray;
    unsigned int dstPitch;

  unsigned int WidthInBytes;
  unsigned int Height;
} CUDA_MEMCPY2D;
```

where:

- srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
  CU_MEMORYTYPE_HOST = 0x01,
  CU_MEMORYTYPE_DEVICE = 0x02,
  CU_MEMORYTYPE_ARRAY = 0x03,
  CU_MEMORYTYPE_UNIFIED = 0x04
} CUmemorytype;
```

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_UNIFIED**, srcDevice and srcPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. srcArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_HOST**, srcHost and srcPitch specify the (host) base address of the source data and the bytes per row to apply. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_DEVICE**, srcDevice and srcPitch specify the (device) base address of the source data and the bytes per row to apply. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_ARRAY**, srcArray specifies the handle of the source data. srcHost, srcDevice and srcPitch are ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_HOST**, dstHost and dstPitch specify the (host) base address of the destination data and the bytes per row to apply. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_UNIFIED**, dstDevice and dstPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. dstArray



is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_DEVICE**, dstDevice and dstPitch specify the (device) base address of the destination data and the bytes per row to apply. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_ARRAY**, dstArray specifies the handle of the destination data. dstHost, dstDevice and dstPitch are ignored.

- srcXInBytes and srcY specify the base address of the source data for the copy.

.RS 4 For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+srcY*srcPitch + srcXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+srcY*srcPitch+srcXInBytes;
```

.RS 4 For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

- dstXInBytes and dstY specify the base address of the destination data for the copy.

.RS 4 For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+dstY*dstPitch + dstXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+dstY*dstPitch+dstXInBytes;
```

.RS 4 For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

- WidthInBytes and Height specify the width (in bytes) and height of the 2D copy being performed.
- If specified, srcPitch must be greater than or equal to WidthInBytes + srcXInBytes, and dstPitch must be greater than or equal to WidthInBytes + dstXInBytes.

.RS 4 **cuMemcpy2D()** returns an error if any pitch is greater than the maximum allowed (**CU_DEVICE_ATTRIBUTE_MAX_PITCH**). **cuMemAllocPitch()** passes back pitches that always work with **cuMemcpy2D()**. On intra-device memory copies (device to device, CUDA array to device, CUDA array to CUDA array), **cuMemcpy2D()** may fail for pitches not computed by **cuMemAllocPitch()**. **cuMemcpy2DUnaligned()** does not have this restriction, but may run significantly slower in the cases where **cuMemcpy2D()** would have returned an error code.

Parameters:

pCopy - Parameters for the memory copy

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**



CUresult cuMemcpy2DAsync (const CUDA_MEMCPY2D * pCopy, CUstream hStream)

Perform a 2D memory copy according to the parameters specified in `pCopy`. The **CUDA_MEMCPY2D** structure is defined as:

```
typedef struct CUDA_MEMCPY2D_st {
  unsigned int srcXInBytes, srcY;
  CUmemorytype srcMemoryType;
  const void *srcHost;
  CUdeviceptr srcDevice;
  CUarray srcArray;
  unsigned int srcPitch;
  unsigned int dstXInBytes, dstY;
  CUmemorytype dstMemoryType;
  void *dstHost;
  CUdeviceptr dstDevice;
  CUarray dstArray;
  unsigned int dstPitch;
  unsigned int WidthInBytes;
  unsigned int Height;
} CUDA_MEMCPY2D;
```

where:

- srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
  CU_MEMORYTYPE_HOST = 0x01,
  CU_MEMORYTYPE_DEVICE = 0x02,
  CU_MEMORYTYPE_ARRAY = 0x03,
  CU_MEMORYTYPE_UNIFIED = 0x04
} CUmemorytype;
```

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_HOST**, srcHost and srcPitch specify the (host) base address of the source data and the bytes per row to apply. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_UNIFIED**, srcDevice and srcPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. srcArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_DEVICE**, srcDevice and srcPitch specify the (device) base address of the source data and the bytes per row to apply. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_ARRAY**, srcArray specifies the handle of the source data. srcHost, srcDevice and srcPitch are ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_UNIFIED**, dstDevice and dstPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. dstArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_HOST**, dstHost and dstPitch specify the (host) base address of the destination data and the bytes per row to apply. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_DEVICE**, dstDevice and dstPitch specify the (device) base address of the destination data and the bytes per row to apply. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_ARRAY**, dstArray specifies the handle of the destination data. dstHost, dstDevice and dstPitch are ignored.

- srcXInBytes and srcY specify the base address of the source data for the copy.

.RS 4 For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+srcY*srcPitch + srcXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+srcY*srcPitch+srcXInBytes;
```



.RS 4 For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

- dstXInBytes and dstY specify the base address of the destination data for the copy.

.RS 4 For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+dstY*dstPitch + dstXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+dstY*dstPitch+dstXInBytes;
```

.RS 4 For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

- WidthInBytes and Height specify the width (in bytes) and height of the 2D copy being performed.
- If specified, srcPitch must be greater than or equal to WidthInBytes + srcXInBytes, and dstPitch must be greater than or equal to WidthInBytes + dstXInBytes.
- If specified, srcPitch must be greater than or equal to WidthInBytes + srcXInBytes, and dstPitch must be greater than or equal to WidthInBytes + dstXInBytes.
- If specified, srcHeight must be greater than or equal to Height + srcY, and dstHeight must be greater than or equal to Height + dstY.

.RS 4 **cuMemcpy2DAsync()** returns an error if any pitch is greater than the maximum allowed (**CU_DEVICE_ATTRIBUTE_MAX_PITCH**). **cuMemAllocPitch()** passes back pitches that always work with **cuMemcpy2D()**. On intra-device memory copies (device to device, CUDA array to device, CUDA array to CUDA array), **cuMemcpy2DAsync()** may fail for pitches not computed by **cuMemAllocPitch()**.

Parameters:

pCopy - Parameters for the memory copy
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemcpy2DUnaligned (const CUDA_MEMCPY2D * pCopy)

Perform a 2D memory copy according to the parameters specified in `pCopy`. The **CUDA_MEMCPY2D** structure is defined as:

```
typedef struct CUDA_MEMCPY2D_st {
  unsigned int srcXInBytes, srcY;
  CUmemorytype srcMemoryType;
  const void *srcHost;
```



```
    CUdeviceptr srcDevice;
    CUarray srcArray;
    unsigned int srcPitch;
    unsigned int dstXInBytes, dstY;
    CUmemorytype dstMemoryType;
    void *dstHost;
    CUdeviceptr dstDevice;
    CUarray dstArray;
    unsigned int dstPitch;
    unsigned int WidthInBytes;
    unsigned int Height;
  } CUDA_MEMCPY2D;
```

where:

- srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
  CU_MEMORYTYPE_HOST = 0x01,
  CU_MEMORYTYPE_DEVICE = 0x02,
  CU_MEMORYTYPE_ARRAY = 0x03,
  CU_MEMORYTYPE_UNIFIED = 0x04
} CUmemorytype;
```

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_UNIFIED**, srcDevice and srcPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. srcArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_HOST**, srcHost and srcPitch specify the (host) base address of the source data and the bytes per row to apply. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_DEVICE**, srcDevice and srcPitch specify the (device) base address of the source data and the bytes per row to apply. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_ARRAY**, srcArray specifies the handle of the source data. srcHost, srcDevice and srcPitch are ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_UNIFIED**, dstDevice and dstPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. dstArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_HOST**, dstHost and dstPitch specify the (host) base address of the destination data and the bytes per row to apply. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_DEVICE**, dstDevice and dstPitch specify the (device) base address of the destination data and the bytes per row to apply. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_ARRAY**, dstArray specifies the handle of the destination data. dstHost, dstDevice and dstPitch are ignored.

- srcXInBytes and srcY specify the base address of the source data for the copy.

.RS 4 For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+srcY*srcPitch + srcXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+srcY*srcPitch+srcXInBytes;
```

.RS 4 For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

- dstXInBytes and dstY specify the base address of the destination data for the copy.

.RS 4 For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+dstY*dstPitch + dstXInBytes);
```



.RS 4 For device pointers, the starting address is

CUdeviceptr dstStart = dstDevice+dstY*dstPitch+dstXInBytes;

.RS 4 For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

- WidthInBytes and Height specify the width (in bytes) and height of the 2D copy being performed.
- If specified, srcPitch must be greater than or equal to WidthInBytes + srcXInBytes, and dstPitch must be greater than or equal to WidthInBytes + dstXInBytes.

.RS 4 **cuMemcpy2D()** returns an error if any pitch is greater than the maximum allowed (**CU_DEVICE_ATTRIBUTE_MAX_PITCH**). **cuMemAllocPitch()** passes back pitches that always work with **cuMemcpy2D()**. On intra-device memory copies (device to device, CUDA array to device, CUDA array to CUDA array), **cuMemcpy2D()** may fail for pitches not computed by **cuMemAllocPitch()**. **cuMemcpy2DUnaligned()** does not have this restriction, but may run significantly slower in the cases where **cuMemcpy2D()** would have returned an error code.

Parameters:

pCopy - Parameters for the memory copy

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpy3D (const CUDA_MEMCPY3D * pCopy)

Perform a 3D memory copy according to the parameters specified in `pCopy`. The **CUDA_MEMCPY3D** structure is defined as:

```
typedef struct CUDA_MEMCPY3D_st {

    unsigned int srcXInBytes, srcY, srcZ;
    unsigned int srcLOD;
    CUmemorytype srcMemoryType;
        const void *srcHost;
        CUdeviceptr srcDevice;
        CUarray srcArray;
        unsigned int srcPitch;  // ignored when src is array
        unsigned int srcHeight; // ignored when src is array; may be 0 if Depth==1

    unsigned int dstXInBytes, dstY, dstZ;
    unsigned int dstLOD;
    CUmemorytype dstMemoryType;
        void *dstHost;
        CUdeviceptr dstDevice;
        CUarray dstArray;
        unsigned int dstPitch;  // ignored when dst is array
        unsigned int dstHeight; // ignored when dst is array; may be 0 if Depth==1
```



```
    unsigned int WidthInBytes;
    unsigned int Height;
    unsigned int Depth;
} CUDA_MEMCPY3D;
```

where:

- srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
  CU_MEMORYTYPE_HOST = 0x01,
  CU_MEMORYTYPE_DEVICE = 0x02,
  CU_MEMORYTYPE_ARRAY = 0x03,
  CU_MEMORYTYPE_UNIFIED = 0x04
} CUmemorytype;
```

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_UNIFIED**, srcDevice and srcPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. srcArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_HOST**, srcHost, srcPitch and srcHeight specify the (host) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_DEVICE**, srcDevice, srcPitch and srcHeight specify the (device) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_ARRAY**, srcArray specifies the handle of the source data. srcHost, srcDevice, srcPitch and srcHeight are ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_UNIFIED**, dstDevice and dstPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. dstArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_HOST**, dstHost and dstPitch specify the (host) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_DEVICE**, dstDevice and dstPitch specify the (device) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_ARRAY**, dstArray specifies the handle of the destination data. dstHost, dstDevice, dstPitch and dstHeight are ignored.

- srcXInBytes, srcY and srcZ specify the base address of the source data for the copy.

.RS 4 For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+(srcZ*srcHeight+srcY)*srcPitch + srcXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+(srcZ*srcHeight+srcY)*srcPitch+srcXInBytes;
```

.RS 4 For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

- dstXInBytes, dstY and dstZ specify the base address of the destination data for the copy.

.RS 4 For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+(dstZ*dstHeight+dstY)*dstPitch + dstXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+(dstZ*dstHeight+dstY)*dstPitch+dstXInBytes;
```



.RS 4 For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.

- WidthInBytes, Height and Depth specify the width (in bytes), height and depth of the 3D copy being performed.
- If specified, srcPitch must be greater than or equal to WidthInBytes + srcXInBytes, and dstPitch must be greater than or equal to WidthInBytes + dstXInBytes.
- If specified, srcHeight must be greater than or equal to Height + srcY, and dstHeight must be greater than or equal to Height + dstY.

.RS 4 **cuMemcpy3D()** returns an error if any pitch is greater than the maximum allowed (**CU_DEVICE_ATTRIBUTE_MAX_PITCH**).

The srcLOD and dstLOD members of the **CUDA_MEMCPY3D** structure must be set to 0.

Parameters:

pCopy - Parameters for the memory copy

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpy3DAsync (const CUDA_MEMCPY3D * pCopy, CUstream hStream)

Perform a 3D memory copy according to the parameters specified in `pCopy`. The **CUDA_MEMCPY3D** structure is defined as:

```
typedef struct CUDA_MEMCPY3D_st {

    unsigned int srcXInBytes, srcY, srcZ;
    unsigned int srcLOD;
    CUmemorytype srcMemoryType;
        const void *srcHost;
        CUdeviceptr srcDevice;
        CUarray srcArray;
        unsigned int srcPitch;  // ignored when src is array
        unsigned int srcHeight; // ignored when src is array; may be 0 if Depth==1

    unsigned int dstXInBytes, dstY, dstZ;
    unsigned int dstLOD;
    CUmemorytype dstMemoryType;
        void *dstHost;
        CUdeviceptr dstDevice;
        CUarray dstArray;
        unsigned int dstPitch;  // ignored when dst is array
        unsigned int dstHeight; // ignored when dst is array; may be 0 if Depth==1

    unsigned int WidthInBytes;
    unsigned int Height;
```



```
    unsigned int Depth;
} CUDA_MEMCPY3D;
```

where:

- srcMemoryType and dstMemoryType specify the type of memory of the source and destination, respectively; CUmemorytype_enum is defined as:

```
typedef enum CUmemorytype_enum {
  CU_MEMORYTYPE_HOST = 0x01,
  CU_MEMORYTYPE_DEVICE = 0x02,
  CU_MEMORYTYPE_ARRAY = 0x03,
  CU_MEMORYTYPE_UNIFIED = 0x04
} CUmemorytype;
```

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_UNIFIED**, srcDevice and srcPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. srcArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_HOST**, srcHost, srcPitch and srcHeight specify the (host) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_DEVICE**, srcDevice, srcPitch and srcHeight specify the (device) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. srcArray is ignored.

.RS 4 If srcMemoryType is **CU_MEMORYTYPE_ARRAY**, srcArray specifies the handle of the source data. srcHost, srcDevice, srcPitch and srcHeight are ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_UNIFIED**, dstDevice and dstPitch specify the (unified virtual address space) base address of the source data and the bytes per row to apply. dstArray is ignored. This value may be used only if unified addressing is supported in the calling context.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_HOST**, dstHost and dstPitch specify the (host) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_DEVICE**, dstDevice and dstPitch specify the (device) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. dstArray is ignored.

.RS 4 If dstMemoryType is **CU_MEMORYTYPE_ARRAY**, dstArray specifies the handle of the destination data. dstHost, dstDevice, dstPitch and dstHeight are ignored.

- srcXInBytes, srcY and srcZ specify the base address of the source data for the copy.

.RS 4 For host pointers, the starting address is

```
void* Start = (void*)((char*)srcHost+(srcZ*srcHeight+srcY)*srcPitch + srcXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr Start = srcDevice+(srcZ*srcHeight+srcY)*srcPitch+srcXInBytes;
```

.RS 4 For CUDA arrays, srcXInBytes must be evenly divisible by the array element size.

- dstXInBytes, dstY and dstZ specify the base address of the destination data for the copy.

.RS 4 For host pointers, the base address is

```
void* dstStart = (void*)((char*)dstHost+(dstZ*dstHeight+dstY)*dstPitch + dstXInBytes);
```

.RS 4 For device pointers, the starting address is

```
CUdeviceptr dstStart = dstDevice+(dstZ*dstHeight+dstY)*dstPitch+dstXInBytes;
```

.RS 4 For CUDA arrays, dstXInBytes must be evenly divisible by the array element size.



- WidthInBytes, Height and Depth specify the width (in bytes), height and depth of the 3D copy being performed.
- If specified, srcPitch must be greater than or equal to WidthInBytes + srcXInBytes, and dstPitch must be greater than or equal to WidthInBytes + dstXInBytes.
- If specified, srcHeight must be greater than or equal to Height + srcY, and dstHeight must be greater than or equal to Height + dstY.

.RS 4 **cuMemcpy3DAsync()** returns an error if any pitch is greater than the maximum allowed (**CU_DEVICE_ATTRIBUTE_MAX_PITCH**).

The srcLOD and dstLOD members of the **CUDA_MEMCPY3D** structure must be set to 0.

Parameters:

pCopy - Parameters for the memory copy
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemcpy3DPeer (const CUDA_MEMCPY3D_PEER * pCopy)

Perform a 3D memory copy according to the parameters specified in `pCopy`. See the definition of the **CUDA_MEMCPY3D_PEER** structure for documentation of its parameters.

Parameters:

pCopy - Parameters for the memory copy

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuMemcpyDtoD, **cuMemcpyPeer**, **cuMemcpyDtoDAsync**, **cuMemcpyPeerAsync**, **cuMemcpy3DPeerAsync**

CUresult cuMemcpy3DPeerAsync (const CUDA_MEMCPY3D_PEER * pCopy, CUstream hStream)

Perform a 3D memory copy according to the parameters specified in `pCopy`. See the definition of the **CUDA_MEMCPY3D_PEER** structure for documentation of its parameters.

Parameters:



pCopy - Parameters for the memory copy
hStream - Stream identifier

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:
cuMemcpyDtoD, **cuMemcpyPeer**, **cuMemcpyDtoDAsync**, **cuMemcpyPeerAsync**, **cuMemcpy3DPeerAsync**

CUresult cuMemcpyAsync (CUdeviceptr dst, CUdeviceptr src, size_t ByteCount, CUstream hStream)
Copies data between two pointers. `dst` and `src` are base pointers of the destination and source, respectively. `ByteCount` specifies the number of bytes to copy. Note that this function infers the type of the transfer (host to host, host to device, device to device, or device to host) from the pointer values. This function is only allowed in contexts which support unified addressing.

Parameters:
dst - Destination unified virtual address space pointer
src - Source unified virtual address space pointer
ByteCount - Size of memory copy in bytes
hStream - Stream identifier

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:
cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemcpyAtoA (CUarray dstArray, size_t dstOffset, CUarray srcArray, size_t srcOffset, size_t ByteCount)
Copies from one 1D CUDA array to another. `dstArray` and `srcArray` specify the handles of the destination and source CUDA arrays for the copy, respectively. `dstOffset` and `srcOffset` specify the destination and source offsets in bytes into the CUDA arrays. `ByteCount` is the number of bytes to be copied. The size of the elements in the CUDA arrays need not be the same format, but the elements must be the same size; and count must be evenly divisible by that size.

Parameters:
dstArray - Destination array
dstOffset - Offset in bytes of destination array



srcArray - Source array
srcOffset - Offset in bytes of source array
ByteCount - Size of memory copy in bytes

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:
cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpyAtoD (CUdeviceptr dstDevice, CUarray srcArray, size_t srcOffset, size_t ByteCount)
Copies from one 1D CUDA array to device memory. `dstDevice` specifies the base pointer of the destination and must be naturally aligned with the CUDA array elements. `srcArray` and `srcOffset` specify the CUDA array handle and the offset in bytes into the array where the copy is to begin. `ByteCount` specifies the number of bytes to copy and must be evenly divisible by the array element size.

Parameters:
dstDevice - Destination device pointer
srcArray - Source array
srcOffset - Offset in bytes of source array
ByteCount - Size of memory copy in bytes

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:
cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpyAtoH (void * dstHost, CUarray srcArray, size_t srcOffset, size_t ByteCount)
Copies from one 1D CUDA array to host memory. `dstHost` specifies the base pointer of the destination. `srcArray` and `srcOffset` specify the CUDA array handle and starting offset in bytes of the source data. `ByteCount` specifies the number of bytes to copy.

Parameters:



dstHost - Destination device pointer
srcArray - Source array
srcOffset - Offset in bytes of source array
ByteCount - Size of memory copy in bytes

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:
cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpyAtoHAsync (void * dstHost, CUarray srcArray, size_t srcOffset, size_t ByteCount, CUstream hStream)
Copies from one 1D CUDA array to host memory. `dstHost` specifies the base pointer of the destination. `srcArray` and `srcOffset` specify the CUDA array handle and starting offset in bytes of the source data. `ByteCount` specifies the number of bytes to copy.

Parameters:
dstHost - Destination pointer
srcArray - Source array
srcOffset - Offset in bytes of source array
ByteCount - Size of memory copy in bytes
hStream - Stream identifier

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:
cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**



CUresult cuMemcpyDtoA (CUarray dstArray, size_t dstOffset, CUdeviceptr srcDevice, size_t ByteCount)

Copies from device memory to a 1D CUDA array. `dstArray` and `dstOffset` specify the CUDA array handle and starting index of the destination data. `srcDevice` specifies the base pointer of the source. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstArray - Destination array
dstOffset - Offset in bytes of destination array
srcDevice - Source device pointer
ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpyDtoD (CUdeviceptr dstDevice, CUdeviceptr srcDevice, size_t ByteCount)

Copies from device memory to device memory. `dstDevice` and `srcDevice` are the base pointers of the destination and source, respectively. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstDevice - Destination device pointer
srcDevice - Source device pointer
ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpyDtoDAsync (CUdeviceptr dstDevice, CUdeviceptr srcDevice, size_t ByteCount, CUstream hStream)

Copies from device memory to device memory. `dstDevice` and `srcDevice` are the base pointers



of the destination and source, respectively. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstDevice - Destination device pointer
srcDevice - Source device pointer
ByteCount - Size of memory copy in bytes
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemcpyDtoH (void * dstHost, CUdeviceptr srcDevice, size_t ByteCount)

Copies from device to host memory. `dstHost` and `srcDevice` specify the base pointers of the destination and source, respectively. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstHost - Destination host pointer
srcDevice - Source device pointer
ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpyDtoHAsync (void * dstHost, CUdeviceptr srcDevice, size_t ByteCount, CUstream hStream)

Copies from device to host memory. `dstHost` and `srcDevice` specify the base pointers of the



destination and source, respectively. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstHost - Destination host pointer
srcDevice - Source device pointer
ByteCount - Size of memory copy in bytes
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemcpyHtoA (CUarray dstArray, size_t dstOffset, const void * srcHost, size_t ByteCount)

Copies from host memory to a 1D CUDA array. `dstArray` and `dstOffset` specify the CUDA array handle and starting offset in bytes of the destination data. `pSrc` specifies the base address of the source. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstArray - Destination array
dstOffset - Offset in bytes of destination array
srcHost - Source host pointer
ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**



CUresult cuMemcpyHtoAAsync (CUarray dstArray, size_t dstOffset, const void * srcHost, size_t ByteCount, CUstream hStream)

Copies from host memory to a 1D CUDA array. `dstArray` and `dstOffset` specify the CUDA array handle and starting offset in bytes of the destination data. `srcHost` specifies the base address of the source. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstArray - Destination array
dstOffset - Offset in bytes of destination array
srcHost - Source host pointer
ByteCount - Size of memory copy in bytes
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemcpyHtoD (CUdeviceptr dstDevice, const void * srcHost, size_t ByteCount)

Copies from host memory to device memory. `dstDevice` and `srcHost` are the base addresses of the destination and source, respectively. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstDevice - Destination device pointer
srcHost - Source host pointer
ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**,



cuMemsetD8, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemcpyHtoDAsync (CUdeviceptr dstDevice, const void * srcHost, size_t ByteCount, CUstream hStream)

Copies from host memory to device memory. `dstDevice` and `srcHost` are the base addresses of the destination and source, respectively. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstDevice - Destination device pointer
srcHost - Source host pointer
ByteCount - Size of memory copy in bytes
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemcpyPeer (CUdeviceptr dstDevice, CUcontext dstContext, CUdeviceptr srcDevice, CUcontext srcContext, size_t ByteCount)

Copies from device memory in one context to device memory in another context. `dstDevice` is the base device pointer of the destination memory and `dstContext` is the destination context. `srcDevice` is the base device pointer of the source memory and `srcContext` is the source pointer. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstDevice - Destination device pointer
dstContext - Destination context
srcDevice - Source device pointer
srcContext - Source context
ByteCount - Size of memory copy in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

See also:

cuMemcpyDtoD, **cuMemcpy3DPeer**, **cuMemcpyDtoDAsync**, **cuMemcpyPeerAsync**, **cuMemcpy3DPeerAsync**



CUresult cuMemcpyPeerAsync (CUdeviceptr dstDevice, CUcontext dstContext, CUdeviceptr srcDevice, CUcontext srcContext, size_t ByteCount, CUstream hStream)

Copies from device memory in one context to device memory in another context. `dstDevice` is the base device pointer of the destination memory and `dstContext` is the destination context. `srcDevice` is the base device pointer of the source memory and `srcContext` is the source pointer. `ByteCount` specifies the number of bytes to copy.

Parameters:

dstDevice - Destination device pointer
dstContext - Destination context
srcDevice - Source device pointer
srcContext - Source context
ByteCount - Size of memory copy in bytes
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cuMemcpyDtoD, **cuMemcpyPeer**, **cuMemcpy3DPeer**, **cuMemcpyDtoDAsync**, **cuMemcpy3DPeerAsync**

CUresult cuMemFree (CUdeviceptr dptr)

Frees the memory space pointed to by `dptr`, which must have been returned by a previous call to **cuMemAlloc()** or **cuMemAllocPitch()**.

Parameters:

dptr - Pointer to memory to free

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemFreeHost (void * p)

Frees the memory space pointed to by `p`, which must have been returned by a previous call to **cuMemAllocHost()**.

Parameters:

p - Pointer to memory to free

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**,



CUDA_ERROR_INVALID_VALUE

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemGetAddressRange (CUdeviceptr * pbase, size_t * psize, CUdeviceptr dptr)

Returns the base address in `*pbase` and size in `*psize` of the allocation by **cuMemAlloc()** or **cuMemAllocPitch()** that contains the input pointer `dptr`. Both parameters `pbase` and `psize` are optional. If one of them is NULL, it is ignored.

Parameters:

pbase - Returned base address
psize - Returned size of device memory allocation
dptr - Device pointer to query

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemGetInfo (size_t * free, size_t * total)

Returns in `*free` and `*total` respectively, the free and total amount of memory available for allocation by the CUDA context, in bytes.

Parameters:

free - Returned free memory in bytes
total - Returned total memory in bytes

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**,



cuMemcpyDtoA, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemHostAlloc (void ** pp, size_t bytesize, unsigned int Flags)

Allocates `bytesize` bytes of host memory that is page-locked and accessible to the device. The driver tracks the virtual memory ranges allocated with this function and automatically accelerates calls to functions such as **cuMemcpyHtoD()**. Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as malloc(). Allocating excessive amounts of pinned memory may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to allocate staging areas for data exchange between host and device.

The `Flags` parameter enables different options to be specified that affect the allocation, as follows.

- **CU_MEMHOSTALLOC_PORTABLE**: The memory returned by this call will be considered as pinned memory by all CUDA contexts, not just the one that performed the allocation.
- **CU_MEMHOSTALLOC_DEVICEMAP**: Maps the allocation into the CUDA address space. The device pointer to the memory may be obtained by calling **cuMemHostGetDevicePointer()**. This feature is available only on GPUs with compute capability greater than or equal to 1.1.
- **CU_MEMHOSTALLOC_WRITECOMBINED**: Allocates the memory as write-combined (WC). WC memory can be transferred across the PCI Express bus more quickly on some system configurations, but cannot be read efficiently by most CPUs. WC memory is a good option for buffers that will be written by the CPU and read by the GPU via mapped pinned memory or host->device transfers.

All of these flags are orthogonal to one another: a developer may allocate memory that is portable, mapped and/or write-combined with no restrictions.

The CUDA context must have been created with the **CU_CTX_MAP_HOST** flag in order for the **CU_MEMHOSTALLOC_DEVICEMAP** flag to have any effect.

The **CU_MEMHOSTALLOC_DEVICEMAP** flag may be specified on CUDA contexts for devices that do not support mapped pinned memory. The failure is deferred to **cuMemHostGetDevicePointer()** because the memory may be mapped into other CUDA contexts via the **CU_MEMHOSTALLOC_PORTABLE** flag.

The memory allocated by this function must be freed with **cuMemFreeHost()**.

Note all host memory allocated using **cuMemHostAlloc()** will automatically be immediately accessible to all contexts on all devices which support unified addressing (as may be queried using **CU_DEVICE_ATTRIBUTE_UNIFIED_ADDRESSING**). Unless the flag **CU_MEMHOSTALLOC_WRITECOMBINED** is specified, the device pointer that may be used to access this host memory from those contexts is always equal to the returned host pointer `*pp`. If the flag **CU_MEMHOSTALLOC_WRITECOMBINED** is specified, then the function **cuMemHostGetDevicePointer()** must be used to query the device pointer, even if the context supports unified addressing. See **Unified Addressing** for additional details.

Parameters:

pp - Returned host pointer to page-locked memory
bytesize - Requested allocation size in bytes
Flags - Flags for allocation request

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**,



cuArrayGetDescriptor, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemHostGetDevicePointer (CUdeviceptr * pdptr, void * p, unsigned int Flags)

Passes back the device pointer `pdptr` corresponding to the mapped, pinned host buffer `p` allocated by **cuMemHostAlloc**.

cuMemHostGetDevicePointer() will fail if the **CU_MEMHOSTALLOC_DEVICEMAP** flag was not specified at the time the memory was allocated, or if the function is called on a GPU that does not support mapped pinned memory.

`Flags` provides for future releases. For now, it must be set to 0.

Parameters:

pdptr - Returned device pointer
p - Host pointer
Flags - Options (must be 0)

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD32**

CUresult cuMemHostGetFlags (unsigned int * pFlags, void * p)

Passes back the flags `pFlags` that were specified when allocating the pinned host buffer `p` allocated by **cuMemHostAlloc**.

cuMemHostGetFlags() will fail if the pointer does not reside in an allocation performed by **cuMemAllocHost()** or **cuMemHostAlloc()**.

Parameters:

pFlags - Returned flags word
p - Host pointer

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuMemAllocHost, **cuMemHostAlloc**



CUresult cuMemHostRegister (void * p, size_t bytesize, unsigned int Flags)

Page-locks the memory range specified by `p` and `bytesize` and maps it for the device(s) as specified by `Flags`. This memory range also is added to the same tracking mechanism as **cuMemHostAlloc** to automatically accelerate calls to functions such as **cuMemcpyHtoD()**. Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory that has not been registered. Page-locking excessive amounts of memory may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to register staging areas for data exchange between host and device.

This function has limited support on Mac OS X. OS 10.7 or higher is required.

The `Flags` parameter enables different options to be specified that affect the allocation, as follows.

- **CU_MEMHOSTREGISTER_PORTABLE**: The memory returned by this call will be considered as pinned memory by all CUDA contexts, not just the one that performed the allocation.
- **CU_MEMHOSTREGISTER_DEVICEMAP**: Maps the allocation into the CUDA address space. The device pointer to the memory may be obtained by calling **cuMemHostGetDevicePointer()**. This feature is available only on GPUs with compute capability greater than or equal to 1.1.

All of these flags are orthogonal to one another: a developer may page-lock memory that is portable or mapped with no restrictions.

The CUDA context must have been created with the **CU_CTX_MAP_HOST** flag in order for the **CU_MEMHOSTREGISTER_DEVICEMAP** flag to have any effect.

The **CU_MEMHOSTREGISTER_DEVICEMAP** flag may be specified on CUDA contexts for devices that do not support mapped pinned memory. The failure is deferred to **cuMemHostGetDevicePointer()** because the memory may be mapped into other CUDA contexts via the **CU_MEMHOSTREGISTER_PORTABLE** flag.

The memory page-locked by this function must be unregistered with **cuMemHostUnregister()**.

Parameters:

p - Host pointer to memory to page-lock
bytesize - Size in bytes of the address range to page-lock
Flags - Flags for allocation request

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**, **CUDA_ERROR_HOST_MEMORY_ALREADY_REGISTERED**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuMemHostUnregister, **cuMemHostGetFlags**, **cuMemHostGetDevicePointer**

CUresult cuMemHostUnregister (void * p)

Unmaps the memory range whose base address is specified by `p`, and makes it pageable again.

The base address must be the same one specified to **cuMemHostRegister()**.

Parameters:

p - Host pointer to memory to unregister

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**, **CUDA_ERROR_HOST_MEMORY_NOT_REGISTERED**,

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuMemHostRegister



CUresult cuMemsetD16 (CUdeviceptr dstDevice, unsigned short us, size_t N)

Sets the memory range of `N` 16-bit values to the specified value `us`. The `dstDevice` pointer must be two byte aligned.

Parameters:

dstDevice - Destination device pointer
us - Value to set
N - Number of elements

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemsetD16Async (CUdeviceptr dstDevice, unsigned short us, size_t N, CUstream hStream)

Sets the memory range of `N` 16-bit values to the specified value `us`. The `dstDevice` pointer must be two byte aligned.

Parameters:

dstDevice - Destination device pointer
us - Value to set
N - Number of elements
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD32**,



cuMemsetD32Async

CUresult cuMemsetD2D16 (CUdeviceptr dstDevice, size_t dstPitch, unsigned short us, size_t Width, size_t Height)

Sets the 2D memory range of `Width` 16-bit values to the specified value `us`. `Height` specifies the number of rows to set, and `dstPitch` specifies the number of bytes between each row. The `dstDevice` pointer and `dstPitch` offset must be two byte aligned. This function performs fastest when the pitch is one that has been passed back by **cuMemAllocPitch()**.

Parameters:

dstDevice - Destination device pointer
dstPitch - Pitch of destination device pointer
us - Value to set
Width - Width of row
Height - Number of rows

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemsetD2D16Async (CUdeviceptr dstDevice, size_t dstPitch, unsigned short us, size_t Width, size_t Height, CUstream hStream)

Sets the 2D memory range of `Width` 16-bit values to the specified value `us`. `Height` specifies the number of rows to set, and `dstPitch` specifies the number of bytes between each row. The `dstDevice` pointer and `dstPitch` offset must be two byte aligned. This function performs fastest when the pitch is one that has been passed back by **cuMemAllocPitch()**.

Parameters:

dstDevice - Destination device pointer
dstPitch - Pitch of destination device pointer
us - Value to set
Width - Width of row
Height - Number of rows
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

This function uses standard semantics.



See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemsetD2D32 (CUdeviceptr dstDevice, size_t dstPitch, unsigned int ui, size_t Width, size_t Height)

Sets the 2D memory range of `Width` 32-bit values to the specified value `ui`. `Height` specifies the number of rows to set, and `dstPitch` specifies the number of bytes between each row. The `dstDevice` pointer and `dstPitch` offset must be four byte aligned. This function performs fastest when the pitch is one that has been passed back by **cuMemAllocPitch()**.

Parameters:

dstDevice - Destination device pointer
dstPitch - Pitch of destination device pointer
ui - Value to set
Width - Width of row
Height - Number of rows

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemsetD2D32Async (CUdeviceptr dstDevice, size_t dstPitch, unsigned int ui, size_t Width, size_t Height, CUstream hStream)

Sets the 2D memory range of `Width` 32-bit values to the specified value `ui`. `Height` specifies the number of rows to set, and `dstPitch` specifies the number of bytes between each row. The `dstDevice` pointer and `dstPitch` offset must be four byte aligned. This function performs fastest when the pitch is one that has been passed back by **cuMemAllocPitch()**.

Parameters:

dstDevice - Destination device pointer
dstPitch - Pitch of destination device pointer
ui - Value to set
Width - Width of row
Height - Number of rows



hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemsetD2D8 (CUdeviceptr dstDevice, size_t dstPitch, unsigned char uc, size_t Width, size_t Height)

Sets the 2D memory range of `Width` 8-bit values to the specified value `uc`. `Height` specifies the number of rows to set, and `dstPitch` specifies the number of bytes between each row. This function performs fastest when the pitch is one that has been passed back by **cuMemAllocPitch()**.

Parameters:

dstDevice - Destination device pointer
dstPitch - Pitch of destination device pointer
uc - Value to set
Width - Width of row
Height - Number of rows

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemsetD2D8Async (CUdeviceptr dstDevice, size_t dstPitch, unsigned char uc, size_t Width, size_t Height, CUstream hStream)

Sets the 2D memory range of `Width` 8-bit values to the specified value `uc`. `Height` specifies the



number of rows to set, and `dstPitch` specifies the number of bytes between each row. This function performs fastest when the pitch is one that has been passed back by **cuMemAllocPitch()**.

Parameters:

dstDevice - Destination device pointer
dstPitch - Pitch of destination device pointer
uc - Value to set
Width - Width of row
Height - Number of rows
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMemsetD32 (CUdeviceptr dstDevice, unsigned int ui, size_t N)

Sets the memory range of `N` 32-bit values to the specified value `ui`. The `dstDevice` pointer must be four byte aligned.

Parameters:

dstDevice - Destination device pointer
ui - Value to set
N - Number of elements

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**,



cuMemsetD16Async, **cuMemsetD32Async**

CUresult cuMemsetD32Async (CUdeviceptr dstDevice, unsigned int ui, size_t N, CUstream hStream)

Sets the memory range of `N` 32-bit values to the specified value `ui`. The `dstDevice` pointer must be four byte aligned.

Parameters:

dstDevice - Destination device pointer
ui - Value to set
N - Number of elements
hStream - Stream identifier

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

This function uses standard semantics.

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**

CUresult cuMemsetD8 (CUdeviceptr dstDevice, unsigned char uc, size_t N)

Sets the memory range of `N` 8-bit values to the specified value `uc`.

Parameters:

dstDevice - Destination device pointer
uc - Value to set
N - Number of elements

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also .

See also:

cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8Async**, **cuMemsetD16**, **cuMemsetD16Async**,



cuMemsetD32, **cuMemsetD32Async**

CUresult cuMemsetD8Async (CUdeviceptr dstDevice, unsigned char uc, size_t N, CUstream hStream)
Sets the memory range of `N` 8-bit values to the specified value `uc`.

Parameters:
dstDevice - Destination device pointer
uc - Value to set
N - Number of elements
hStream - Stream identifier

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also .

This function uses standard semantics.

See also:
cuArray3DCreate, **cuArray3DGetDescriptor**, **cuArrayCreate**, **cuArrayDestroy**, **cuArrayGetDescriptor**, **cuMemAlloc**, **cuMemAllocHost**, **cuMemAllocPitch**, **cuMemcpy2D**, **cuMemcpy2DAsync**, **cuMemcpy2DUnaligned**, **cuMemcpy3D**, **cuMemcpy3DAsync**, **cuMemcpyAtoA**, **cuMemcpyAtoD**, **cuMemcpyAtoH**, **cuMemcpyAtoHAsync**, **cuMemcpyDtoA**, **cuMemcpyDtoD**, **cuMemcpyDtoDAsync**, **cuMemcpyDtoH**, **cuMemcpyDtoHAsync**, **cuMemcpyHtoA**, **cuMemcpyHtoAAsync**, **cuMemcpyHtoD**, **cuMemcpyHtoDAsync**, **cuMemFree**, **cuMemFreeHost**, **cuMemGetAddressRange**, **cuMemGetInfo**, **cuMemHostAlloc**, **cuMemHostGetDevicePointer**, **cuMemsetD2D8**, **cuMemsetD2D8Async**, **cuMemsetD2D16**, **cuMemsetD2D16Async**, **cuMemsetD2D32**, **cuMemsetD2D32Async**, **cuMemsetD8**, **cuMemsetD16**, **cuMemsetD16Async**, **cuMemsetD32**, **cuMemsetD32Async**

CUresult cuMipmappedArrayCreate (CUmipmappedArray * pHandle, const CUDA_ARRAY3D_DESCRIPTOR * pMipmappedArrayDesc, unsigned int numMipmapLevels)
Creates a CUDA mipmapped array according to the **CUDA_ARRAY3D_DESCRIPTOR** structure `pMipmappedArrayDesc` and returns a handle to the new CUDA mipmapped array in `*pHandle`. `numMipmapLevels` specifies the number of mipmap levels to be allocated. This value is clamped to the range [1, 1 + floor(log2(max(width, height, depth)))].

The **CUDA_ARRAY3D_DESCRIPTOR** is defined as:

```
typedef struct {
    unsigned int Width;
    unsigned int Height;
    unsigned int Depth;
    CUarray_format Format;
    unsigned int NumChannels;
    unsigned int Flags;
} CUDA_ARRAY3D_DESCRIPTOR;
```

where:

- `Width`, `Height`, and `Depth` are the width, height, and depth of the CUDA array (in elements); the following types of CUDA arrays can be allocated:
  - A 1D mipmapped array is allocated if `Height` and `Depth` extents are both zero.
  - A 2D mipmapped array is allocated if only `Depth` extent is zero.
  - A 3D mipmapped array is allocated if all three extents are non-zero.
  - A 1D layered CUDA mipmapped array is allocated if only `Height` is zero and the **CUDA_ARRAY3D_LAYERED** flag is set. Each layer is a 1D array. The number of layers is determined by the depth extent.



- A 2D layered CUDA mipmapped array is allocated if all three extents are non-zero and the **CUDA_ARRAY3D_LAYERED** flag is set. Each layer is a 2D array. The number of layers is determined by the depth extent.
- A cubemap CUDA mipmapped array is allocated if all three extents are non-zero and the **CUDA_ARRAY3D_CUBEMAP** flag is set. `Width` must be equal to `Height`, and `Depth` must be six. A cubemap is a special type of 2D layered CUDA array, where the six layers represent the six faces of a cube. The order of the six layers in memory is the same as that listed in **CUarray_cubemap_face**.
- A cubemap layered CUDA mipmapped array is allocated if all three extents are non-zero, and both, **CUDA_ARRAY3D_CUBEMAP** and **CUDA_ARRAY3D_LAYERED** flags are set. `Width` must be equal to `Height`, and `Depth` must be a multiple of six. A cubemap layered CUDA array is a special type of 2D layered CUDA array that consists of a collection of cubemaps. The first six layers represent the first cubemap, the next six layers form the second cubemap, and so on.

- Format specifies the format of the elements; **CUarray_format** is defined as:

```
typedef enum CUarray_format_enum {
  CU_AD_FORMAT_UNSIGNED_INT8 = 0x01,
  CU_AD_FORMAT_UNSIGNED_INT16 = 0x02,
  CU_AD_FORMAT_UNSIGNED_INT32 = 0x03,
  CU_AD_FORMAT_SIGNED_INT8 = 0x08,
  CU_AD_FORMAT_SIGNED_INT16 = 0x09,
  CU_AD_FORMAT_SIGNED_INT32 = 0x0a,
  CU_AD_FORMAT_HALF = 0x10,
  CU_AD_FORMAT_FLOAT = 0x20
} CUarray_format;
```

- `NumChannels` specifies the number of packed components per CUDA array element; it may be 1, 2, or 4;
- Flags may be set to
  - **CUDA_ARRAY3D_LAYERED** to enable creation of layered CUDA mipmapped arrays. If this flag is set, `Depth` specifies the number of layers, not the depth of a 3D array.
  - **CUDA_ARRAY3D_SURFACE_LDST** to enable surface references to be bound to individual mipmap levels of the CUDA mipmapped array. If this flag is not set, **cuSurfRefSetArray** will fail when attempting to bind a mipmap level of the CUDA mipmapped array to a surface reference.
  - **CUDA_ARRAY3D_CUBEMAP** to enable creation of mipmapped cubemaps. If this flag is set, `Width` must be equal to `Height`, and `Depth` must be six. If the **CUDA_ARRAY3D_LAYERED** flag is also set, then `Depth` must be a multiple of six.
  - **CUDA_ARRAY3D_TEXTURE_GATHER** to indicate that the CUDA mipmapped array will be used for texture gather. Texture gather can only be performed on 2D CUDA mipmapped arrays.

`Width`, `Height` and `Depth` must meet certain size requirements as listed in the following table. All values are specified in elements. Note that for brevity's sake, the full name of the device attribute is not specified. For ex., TEXTURE1D_MIPMAPPED_WIDTH refers to the device attribute **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_MIPMAPPED_WIDTH**.

CUDA array type Valid extents that must always be met
{(width range in elements), (height range), (depth range)} 1D { (1,TEXTURE1D_MIPMAPPED_WIDTH), 0, 0 } 2D { (1,TEXTURE2D_MIPMAPPED_WIDTH), (1,TEXTURE2D_MIPMAPPED_HEIGHT), 0 } 3D { (1,TEXTURE3D_WIDTH), (1,TEXTURE3D_HEIGHT), (1,TEXTURE3D_DEPTH) }
OR
{ (1,TEXTURE3D_WIDTH_ALTERNATE), (1,TEXTURE3D_HEIGHT_ALTERNATE), (1,TEXTURE3D_DEPTH_ALTERNATE) } 1D Layered { (1,TEXTURE1D_LAYERED_WIDTH), 0, (1,TEXTURE1D_LAYERED_LAYERS) } 2D Layered { (1,TEXTURE2D_LAYERED_WIDTH), (1,TEXTURE2D_LAYERED_HEIGHT), (1,TEXTURE2D_LAYERED_LAYERS) } Cubemap { (1,TEXTURECUBEMAP_WIDTH), (1,TEXTURECUBEMAP_WIDTH), 6 } Cubemap Layered {



(1,TEXTURECUBEMAP_LAYERED_WIDTH), (1,TEXTURECUBEMAP_LAYERED_WIDTH), (1,TEXTURECUBEMAP_LAYERED_LAYERS) }

Parameters:

pHandle - Returned mipmapped array
pMipmappedArrayDesc - mipmapped array descriptor
numMipmapLevels - Number of mipmap levels

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**, **CUDA_ERROR_UNKNOWN**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuMipmappedArrayDestroy, **cuMipmappedArrayGetLevel**, **cuArrayCreate**,

CUresult cuMipmappedArrayDestroy (CUmipmappedArray hMipmappedArray)

Destroys the CUDA mipmapped array `hMipmappedArray`.

Parameters:

hMipmappedArray - Mipmapped array to destroy

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_HANDLE**, **CUDA_ERROR_ARRAY_IS_MAPPED**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuMipmappedArrayCreate, **cuMipmappedArrayGetLevel**, **cuArrayCreate**,

CUresult cuMipmappedArrayGetLevel (CUarray * pLevelArray, CUmipmappedArray hMipmappedArray, unsigned int level)

Returns in `*pLevelArray` a CUDA array that represents a single mipmap level of the CUDA mipmapped array `hMipmappedArray`.

If `level` is greater than the maximum number of levels in this mipmapped array, **CUDA_ERROR_INVALID_VALUE** is returned.

Parameters:

pLevelArray - Returned mipmap level CUDA array
hMipmappedArray - CUDA mipmapped array
level - Mipmap level

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_INVALID_HANDLE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuMipmappedArrayCreate, **cuMipmappedArrayDestroy**, **cuArrayCreate**,

Author

Generated automatically by Doxygen from the source code.

