

NAME

Data types used by CUDA Runtime –

Data Structures

```

struct cudaChannelFormatDesc
struct cudaDeviceProp
struct cudaExtent
struct cudaFuncAttributes
struct cudaIpcEventHandle_t
struct cudaIpcMemHandle_t
struct cudaMemcpy3DParms
struct cudaMemcpy3DPeerParms
struct cudaPitchedPtr
struct cudaPointerAttributes
struct cudaPos
struct cudaResourceDesc
struct cudaResourceViewDesc
struct cudaTextureDesc
struct surfaceReference
struct textureReference

```

Defines

```

#define CUDA_IPC_HANDLE_SIZE 64
#define cudaArrayCubemap 0x04
#define cudaArrayDefault 0x00
#define cudaArrayLayered 0x01
#define cudaArraySurfaceLoadStore 0x02
#define cudaArrayTextureGather 0x08
#define cudaDeviceBlockingSync 0x04
#define cudaDeviceLmemResizeToMax 0x10
#define cudaDeviceMapHost 0x08
#define cudaDeviceMask 0x1f
#define cudaDevicePropDontCare
#define cudaDeviceScheduleAuto 0x00
#define cudaDeviceScheduleBlockingSync 0x04
#define cudaDeviceScheduleMask 0x07
#define cudaDeviceScheduleSpin 0x01
#define cudaDeviceScheduleYield 0x02
#define cudaEventBlockingSync 0x01
#define cudaEventDefault 0x00
#define cudaEventDisableTiming 0x02
#define cudaEventInterprocess 0x04
#define cudaHostAllocDefault 0x00
#define cudaHostAllocMapped 0x02
#define cudaHostAllocPortable 0x01
#define cudaHostAllocWriteCombined 0x04
#define cudaHostRegisterDefault 0x00
#define cudaHostRegisterMapped 0x02
#define cudaHostRegisterPortable 0x01
#define cudaIpcMemLazyEnablePeerAccess 0x01
#define cudaMemAttachGlobal 0x01
#define cudaMemAttachHost 0x02
#define cudaMemAttachSingle 0x04
#define cudaPeerAccessDefault 0x00
#define cudaStreamDefault 0x00
#define cudaStreamNonBlocking 0x01

```

Typedefs

```

typedef struct cudaArray * cudaArray_const_t
typedef struct cudaArray * cudaArray_t

```



```

typedef enum cudaError cudaError_t
typedef struct CUevent_st * cudaEvent_t
typedef struct cudaGraphicsResource * cudaGraphicsResource_t
typedef struct cudaMipmappedArray * cudaMipmappedArray_const_t
typedef struct cudaMipmappedArray * cudaMipmappedArray_t
typedef enum cudaOutputMode cudaOutputMode_t
typedef struct CUstream_st * cudaStream_t
typedef unsigned long long cudaSurfaceObject_t
typedef unsigned long long cudaTextureObject_t
typedef struct CUuuid_st cudaUUID_t

```

Enumerations

```

enum cudaChannelFormatKind { cudaChannelFormatKindSigned = 0,
    cudaChannelFormatKindUnsigned = 1, cudaChannelFormatKindFloat = 2,
    cudaChannelFormatKindNone = 3 }
enum cudaComputeMode { cudaComputeModeDefault = 0, cudaComputeModeExclusive = 1,
    cudaComputeModeProhibited = 2, cudaComputeModeExclusiveProcess = 3 }
enum cudaDeviceAttr { cudaDevAttrMaxThreadsPerBlock = 1, cudaDevAttrMaxBlockDimX =
    2, cudaDevAttrMaxBlockDimY = 3, cudaDevAttrMaxBlockDimZ = 4,
    cudaDevAttrMaxGridDimX = 5, cudaDevAttrMaxGridDimY = 6,
    cudaDevAttrMaxGridDimZ = 7, cudaDevAttrMaxSharedMemoryPerBlock = 8,
    cudaDevAttrTotalConstantMemory = 9, cudaDevAttrWarpSize = 10,
    cudaDevAttrMaxPitch = 11, cudaDevAttrMaxRegistersPerBlock = 12,
    cudaDevAttrClockRate = 13, cudaDevAttrTextureAlignment = 14,
    cudaDevAttrGpuOverlap = 15, cudaDevAttrMultiProcessorCount = 16,
    cudaDevAttrKernelExecTimeout = 17, cudaDevAttrIntegrated = 18,
    cudaDevAttrCanMapHostMemory = 19, cudaDevAttrComputeMode = 20,
    cudaDevAttrMaxTexture1DWidth = 21, cudaDevAttrMaxTexture2DWidth = 22,
    cudaDevAttrMaxTexture2DHeight = 23, cudaDevAttrMaxTexture3DWidth = 24,
    cudaDevAttrMaxTexture3DHeight = 25, cudaDevAttrMaxTexture3DDepth = 26,
    cudaDevAttrMaxTexture2DLayeredWidth = 27,
    cudaDevAttrMaxTexture2DLayeredHeight = 28,
    cudaDevAttrMaxTexture2DLayeredLayers = 29, cudaDevAttrSurfaceAlignment = 30,
    cudaDevAttrConcurrentKernels = 31, cudaDevAttrEccEnabled = 32,
    cudaDevAttrPciBusId = 33, cudaDevAttrPciDeviceId = 34, cudaDevAttrTccDriver = 35,
    cudaDevAttrMemoryClockRate = 36, cudaDevAttrGlobalMemoryBusWidth = 37,
    cudaDevAttrL2CacheSize = 38, cudaDevAttrMaxThreadsPerMultiProcessor = 39,
    cudaDevAttrAsyncEngineCount = 40, cudaDevAttrUnifiedAddressing = 41,
    cudaDevAttrMaxTexture1DLayeredWidth = 42,
    cudaDevAttrMaxTexture1DLayeredLayers = 43,
    cudaDevAttrMaxTexture2DGatherWidth = 45, cudaDevAttrMaxTexture2DGatherHeight
    = 46, cudaDevAttrMaxTexture3DWidthAlt = 47, cudaDevAttrMaxTexture3DHeightAlt =
    48, cudaDevAttrMaxTexture3DDepthAlt = 49, cudaDevAttrPciDomainId = 50,
    cudaDevAttrTexturePitchAlignment = 51, cudaDevAttrMaxTextureCubemapWidth = 52,
    cudaDevAttrMaxTextureCubemapLayeredWidth = 53,
    cudaDevAttrMaxTextureCubemapLayeredLayers = 54,
    cudaDevAttrMaxSurface1DWidth = 55, cudaDevAttrMaxSurface2DWidth = 56,
    cudaDevAttrMaxSurface2DHeight = 57, cudaDevAttrMaxSurface3DWidth = 58,
    cudaDevAttrMaxSurface3DHeight = 59, cudaDevAttrMaxSurface3DDepth = 60,
    cudaDevAttrMaxSurface1DLayeredWidth = 61,
    cudaDevAttrMaxSurface1DLayeredLayers = 62,
    cudaDevAttrMaxSurface2DLayeredWidth = 63,
    cudaDevAttrMaxSurface2DLayeredHeight = 64,
    cudaDevAttrMaxSurface2DLayeredLayers = 65,
    cudaDevAttrMaxSurfaceCubemapWidth = 66,
    cudaDevAttrMaxSurfaceCubemapLayeredWidth = 67,
    cudaDevAttrMaxSurfaceCubemapLayeredLayers = 68,
    cudaDevAttrMaxTexture1DLinearWidth = 69, cudaDevAttrMaxTexture2DLinearWidth
    = 70, cudaDevAttrMaxTexture2DLinearHeight = 71,

```



```

    cudaDevAttrMaxTexture2DLinearPitch = 72,
    cudaDevAttrMaxTexture2DMipmappedWidth = 73,
    cudaDevAttrMaxTexture2DMipmappedHeight = 74,
    cudaDevAttrComputeCapabilityMajor = 75, cudaDevAttrComputeCapabilityMinor =
    76, cudaDevAttrMaxTexture1DMipmappedWidth = 77,
    cudaDevAttrStreamPrioritiesSupported = 78, cudaDevAttrGlobalL1CacheSupported =
    79, cudaDevAttrLocalL1CacheSupported = 80,
    cudaDevAttrMaxSharedMemoryPerMultiprocessor = 81,
    cudaDevAttrMaxRegistersPerMultiprocessor = 82, cudaDevAttrManagedMemory = 83,
    cudaDevAttrIsMultiGpuBoard = 84, cudaDevAttrMultiGpuBoardGroupID = 85 }
enum cudaError { cudaSuccess = 0, cudaErrorMissingConfiguration = 1,
    cudaErrorMemoryAllocation = 2, cudaErrorInitializationError = 3,
    cudaErrorLaunchFailure = 4, cudaErrorPriorLaunchFailure = 5,
    cudaErrorLaunchTimeout = 6, cudaErrorLaunchOutOfResources = 7,
    cudaErrorInvalidDeviceFunction = 8, cudaErrorInvalidConfiguration = 9,
    cudaErrorInvalidDevice = 10, cudaErrorInvalidValue = 11, cudaErrorInvalidPitchValue
    = 12, cudaErrorInvalidSymbol = 13, cudaErrorMapBufferObjectFailed = 14,
    cudaErrorUnmapBufferObjectFailed = 15, cudaErrorInvalidHostPointer = 16,
    cudaErrorInvalidDevicePointer = 17, cudaErrorInvalidTexture = 18,
    cudaErrorInvalidTextureBinding = 19, cudaErrorInvalidChannelDescriptor = 20,
    cudaErrorInvalidMemcpyDirection = 21, cudaErrorAddressOfConstant = 22,
    cudaErrorTextureFetchFailed = 23, cudaErrorTextureNotBound = 24,
    cudaErrorSynchronizationError = 25, cudaErrorInvalidFilterSetting = 26,
    cudaErrorInvalidNormSetting = 27, cudaErrorMixedDeviceExecution = 28,
    cudaErrorCudartUnloading = 29, cudaErrorUnknown = 30,
    cudaErrorNotYetImplemented = 31, cudaErrorMemoryValueTooLarge = 32,
    cudaErrorInvalidResourceHandle = 33, cudaErrorNotReady = 34,
    cudaErrorInsufficientDriver = 35, cudaErrorSetOnActiveProcess = 36,
    cudaErrorInvalidSurface = 37, cudaErrorNoDevice = 38, cudaErrorECCUncorrectable =
    39, cudaErrorSharedObjectSymbolNotFound = 40, cudaErrorSharedObjectInitFailed =
    41, cudaErrorUnsupportedLimit = 42, cudaErrorDuplicateVariableName = 43,
    cudaErrorDuplicateTextureName = 44, cudaErrorDuplicateSurfaceName = 45,
    cudaErrorDevicesUnavailable = 46, cudaErrorInvalidKernelImage = 47,
    cudaErrorNoKernelImageForDevice = 48, cudaErrorIncompatibleDriverContext = 49,
    cudaErrorPeerAccessAlreadyEnabled = 50, cudaErrorPeerAccessNotEnabled = 51,
    cudaErrorDeviceAlreadyInUse = 54, cudaErrorProfilerDisabled = 55,
    cudaErrorProfilerNotInitialized = 56, cudaErrorProfilerAlreadyStarted = 57,
    cudaErrorProfilerAlreadyStopped = 58, cudaErrorAssert = 59, cudaErrorTooManyPeers
    = 60, cudaErrorHostMemoryAlreadyRegistered = 61,
    cudaErrorHostMemoryNotRegistered = 62, cudaErrorOperatingSystem = 63,
    cudaErrorPeerAccessUnsupported = 64, cudaErrorLaunchMaxDepthExceeded = 65,
    cudaErrorLaunchFileScopedTex = 66, cudaErrorLaunchFileScopedSurf = 67,
    cudaErrorSyncDepthExceeded = 68, cudaErrorLaunchPendingCountExceeded = 69,
    cudaErrorNotPermitted = 70, cudaErrorNotSupported = 71,
    cudaErrorHardwareStackError = 72, cudaErrorIllegalInstruction = 73,
    cudaErrorMisalignedAddress = 74, cudaErrorInvalidAddressSpace = 75,
    cudaErrorInvalidPc = 76, cudaErrorIllegalAddress = 77, cudaErrorStartupFailure =
    0x7f, cudaErrorApiFailureBase = 10000 }
enum cudaFuncCache { cudaFuncCachePreferNone = 0, cudaFuncCachePreferShared = 1,
    cudaFuncCachePreferL1 = 2, cudaFuncCachePreferEqual = 3 }
enum cudaGraphicsCubeFace { cudaGraphicsCubeFacePositiveX = 0x00,
    cudaGraphicsCubeFaceNegativeX = 0x01, cudaGraphicsCubeFacePositiveY = 0x02,
    cudaGraphicsCubeFaceNegativeY = 0x03, cudaGraphicsCubeFacePositiveZ = 0x04,
    cudaGraphicsCubeFaceNegativeZ = 0x05 }
enum cudaGraphicsMapFlags { cudaGraphicsMapFlagsNone = 0,
    cudaGraphicsMapFlagsReadOnly = 1, cudaGraphicsMapFlagsWriteDiscard = 2 }
enum cudaGraphicsRegisterFlags { cudaGraphicsRegisterFlagsNone = 0,
    cudaGraphicsRegisterFlagsReadOnly = 1, cudaGraphicsRegisterFlagsWriteDiscard = 2,
    cudaGraphicsRegisterFlagsSurfaceLoadStore = 4,

```



```

    cudaGraphicsRegisterFlagsTextureGather = 8 }
enum cudaLimit { cudaLimitStackSize = 0x00, cudaLimitPrintfFifoSize = 0x01,
    cudaLimitMallocHeapSize = 0x02, cudaLimitDevRuntimeSyncDepth = 0x03,
    cudaLimitDevRuntimePendingLaunchCount = 0x04 }
enum cudaMemcpyKind { cudaMemcpyHostToHost = 0, cudaMemcpyHostToDevice = 1,
    cudaMemcpyDeviceToHost = 2, cudaMemcpyDeviceToDevice = 3, cudaMemcpyDefault =
    4 }
enum cudaMemoryType { cudaMemoryTypeHost = 1, cudaMemoryTypeDevice = 2 }
enum cudaOutputMode { cudaKeyValuePair = 0x00, cudaCSV = 0x01 }
enum cudaResourceType { cudaResourceTypeArray = 0x00,
    cudaResourceTypeMipmappedArray = 0x01, cudaResourceTypeLinear = 0x02,
    cudaResourceTypePitch2D = 0x03 }
enum cudaResourceViewFormat { cudaResViewFormatNone = 0x00,
    cudaResViewFormatUnsignedChar1 = 0x01, cudaResViewFormatUnsignedChar2 =
    0x02, cudaResViewFormatUnsignedChar4 = 0x03, cudaResViewFormatSignedChar1 =
    0x04, cudaResViewFormatSignedChar2 = 0x05, cudaResViewFormatSignedChar4 =
    0x06, cudaResViewFormatUnsignedShort1 = 0x07, cudaResViewFormatUnsignedShort2 =
    0x08, cudaResViewFormatUnsignedShort4 = 0x09, cudaResViewFormatSignedShort1 =
    0x0a, cudaResViewFormatSignedShort2 = 0x0b, cudaResViewFormatSignedShort4 =
    0x0c, cudaResViewFormatUnsignedInt1 = 0x0d, cudaResViewFormatUnsignedInt2 =
    0x0e, cudaResViewFormatUnsignedInt4 = 0x0f, cudaResViewFormatSignedInt1 = 0x10,
    cudaResViewFormatSignedInt2 = 0x11, cudaResViewFormatSignedInt4 = 0x12,
    cudaResViewFormatHalf1 = 0x13, cudaResViewFormatHalf2 = 0x14,
    cudaResViewFormatHalf4 = 0x15, cudaResViewFormatFloat1 = 0x16,
    cudaResViewFormatFloat2 = 0x17, cudaResViewFormatFloat4 = 0x18,
    cudaResViewFormatUnsignedBlockCompressed1 = 0x19,
    cudaResViewFormatUnsignedBlockCompressed2 = 0x1a,
    cudaResViewFormatUnsignedBlockCompressed3 = 0x1b,
    cudaResViewFormatUnsignedBlockCompressed4 = 0x1c,
    cudaResViewFormatSignedBlockCompressed4 = 0x1d,
    cudaResViewFormatUnsignedBlockCompressed5 = 0x1e,
    cudaResViewFormatSignedBlockCompressed5 = 0x1f,
    cudaResViewFormatUnsignedBlockCompressed6H = 0x20,
    cudaResViewFormatSignedBlockCompressed6H = 0x21,
    cudaResViewFormatUnsignedBlockCompressed7 = 0x22 }
enum cudaSharedMemConfig
enum cudaSurfaceBoundaryMode { cudaBoundaryModeZero = 0, cudaBoundaryModeClamp =
    1, cudaBoundaryModeTrap = 2 }
enum cudaSurfaceFormatMode { cudaFormatModeForced = 0, cudaFormatModeAuto = 1 }
enum cudaTextureAddressMode { cudaAddressModeWrap = 0, cudaAddressModeClamp = 1,
    cudaAddressModeMirror = 2, cudaAddressModeBorder = 3 }
enum cudaTextureFilterMode { cudaFilterModePoint = 0, cudaFilterModeLinear = 1 }
enum cudaTextureReadMode { cudaReadModeElementType = 0,
    cudaReadModeNormalizedFloat = 1 }

```

Define Documentation

#define CUDA_IPC_HANDLE_SIZE 64

CUDA IPC Handle Size

#define cudaArrayCubemap 0x04

Must be set in cudaMalloc3DArray to create a cubemap CUDA array

#define cudaArrayDefault 0x00

Default CUDA array allocation flag

#define cudaArrayLayered 0x01

Must be set in cudaMalloc3DArray to create a layered CUDA array

#define cudaArraySurfaceLoadStore 0x02

Must be set in cudaMallocArray or cudaMalloc3DArray in order to bind surfaces to the CUDA array



#define cudaArrayTextureGather 0x08

Must be set in cudaMallocArray or cudaMalloc3DArray in order to perform texture gather operations on the CUDA array

#define cudaDeviceBlockingSync 0x04

Device flag - Use blocking synchronization

Deprecated

This flag was deprecated as of CUDA 4.0 and replaced with **cudaDeviceScheduleBlockingSync**.

#define cudaDeviceLmemResizeToMax 0x10

Device flag - Keep local memory allocation after launch

#define cudaDeviceMapHost 0x08

Device flag - Support mapped pinned allocations

#define cudaDeviceMask 0x1f

Device flags mask

#define cudaDevicePropDontCare

Empty device properties

#define cudaDeviceScheduleAuto 0x00

Device flag - Automatic scheduling

#define cudaDeviceScheduleBlockingSync 0x04

Device flag - Use blocking synchronization

#define cudaDeviceScheduleMask 0x07

Device schedule flags mask

#define cudaDeviceScheduleSpin 0x01

Device flag - Spin default scheduling

#define cudaDeviceScheduleYield 0x02

Device flag - Yield default scheduling

#define cudaEventBlockingSync 0x01

Event uses blocking synchronization

#define cudaEventDefault 0x00

Default event flag

#define cudaEventDisableTiming 0x02

Event will not record timing data

#define cudaEventInterprocess 0x04

Event is suitable for interprocess use. cudaEventDisableTiming must be set

#define cudaHostAllocDefault 0x00

Default page-locked allocation flag

#define cudaHostAllocMapped 0x02

Map allocation into device space

#define cudaHostAllocPortable 0x01

Pinned memory accessible by all CUDA contexts

#define cudaHostAllocWriteCombined 0x04

Write-combined memory

#define cudaHostRegisterDefault 0x00

Default host memory registration flag

#define cudaHostRegisterMapped 0x02

Map registered memory into device space

#define cudaHostRegisterPortable 0x01

Pinned memory accessible by all CUDA contexts



```
#define cudaIpcMemLazyEnablePeerAccess 0x01
    Automatically enable peer access between remote devices as needed

#define cudaMemAttachGlobal 0x01
    Memory can be accessed by any stream on any device

#define cudaMemAttachHost 0x02
    Memory cannot be accessed by any stream on any device

#define cudaMemAttachSingle 0x04
    Memory can only be accessed by a single stream on the associated device

#define cudaPeerAccessDefault 0x00
    Default peer addressing enable flag

#define cudaStreamDefault 0x00
    Default stream flag

#define cudaStreamNonBlocking 0x01
    Stream does not synchronize with stream 0 (the NULL stream)
```

Typedef Documentation

```
typedef struct cudaArray* cudaArray_const_t
    CUDA array (as source copy argument)

typedef struct cudaArray* cudaArray_t
    CUDA array

typedef enum cudaError cudaError_t
    CUDA Error types

typedef struct CUevent_st* cudaEvent_t
    CUDA event types

typedef struct cudaGraphicsResource* cudaGraphicsResource_t
    CUDA graphics resource types

typedef struct cudaMipmappedArray* cudaMipmappedArray_const_t
    CUDA mipmapped array (as source argument)

typedef struct cudaMipmappedArray* cudaMipmappedArray_t
    CUDA mipmapped array

typedef enum cudaOutputMode cudaOutputMode_t
    CUDA output file modes

typedef struct CUstream_st* cudaStream_t
    CUDA stream

typedef unsigned long long cudaSurfaceObject_t
    CUDA Surface object

typedef unsigned long long cudaTextureObject_t
    CUDA texture object

typedef struct CUuuid_st cudaUUID_t
    CUDA UUID types
```

Enumeration Type Documentation

```
enum cudaChannelFormatKind
    Channel format kind

Enumerator:

    cudaChannelFormatKindSigned
        Signed channel format

    cudaChannelFormatKindUnsigned
        Unsigned channel format
```



cudaChannelFormatKindFloat

Float channel format

cudaChannelFormatKindNone

No channel format

enum cudaComputeMode

CUDA device compute modes

Enumerator:

cudaComputeModeDefault

Default compute mode (Multiple threads can use **cudaSetDevice()** with this device)

cudaComputeModeExclusive

Compute-exclusive-thread mode (Only one thread in one process will be able to use **cudaSetDevice()** with this device)

cudaComputeModeProhibited

Compute-prohibited mode (No threads can use **cudaSetDevice()** with this device)

cudaComputeModeExclusiveProcess

Compute-exclusive-process mode (Many threads in one process will be able to use **cudaSetDevice()** with this device)

enum cudaDeviceAttr

CUDA device attributes

Enumerator:

cudaDevAttrMaxThreadsPerBlock

Maximum number of threads per block

cudaDevAttrMaxBlockDimX

Maximum block dimension X

cudaDevAttrMaxBlockDimY

Maximum block dimension Y

cudaDevAttrMaxBlockDimZ

Maximum block dimension Z

cudaDevAttrMaxGridDimX

Maximum grid dimension X

cudaDevAttrMaxGridDimY

Maximum grid dimension Y

cudaDevAttrMaxGridDimZ

Maximum grid dimension Z

cudaDevAttrMaxSharedMemoryPerBlock

Maximum shared memory available per block in bytes

cudaDevAttrTotalConstantMemory

Memory available on device for `__constant__` variables in a CUDA C kernel in bytes

cudaDevAttrWarpSize

Warp size in threads

cudaDevAttrMaxPitch

Maximum pitch in bytes allowed by memory copies

cudaDevAttrMaxRegistersPerBlock

Maximum number of 32-bit registers available per block

cudaDevAttrClockRate

Peak clock frequency in kilohertz

cudaDevAttrTextureAlignment

Alignment requirement for textures



cudaDevAttrGpuOverlap

Device can possibly copy memory and execute a kernel concurrently

cudaDevAttrMultiProcessorCount

Number of multiprocessors on device

cudaDevAttrKernelExecTimeout

Specifies whether there is a run time limit on kernels

cudaDevAttrIntegrated

Device is integrated with host memory

cudaDevAttrCanMapHostMemory

Device can map host memory into CUDA address space

cudaDevAttrComputeMode

Compute mode (See **cudaComputeMode** for details)

cudaDevAttrMaxTexture1DWidth

Maximum 1D texture width

cudaDevAttrMaxTexture2DWidth

Maximum 2D texture width

cudaDevAttrMaxTexture2DHeight

Maximum 2D texture height

cudaDevAttrMaxTexture3DWidth

Maximum 3D texture width

cudaDevAttrMaxTexture3DHeight

Maximum 3D texture height

cudaDevAttrMaxTexture3DDepth

Maximum 3D texture depth

cudaDevAttrMaxTexture2DLayeredWidth

Maximum 2D layered texture width

cudaDevAttrMaxTexture2DLayeredHeight

Maximum 2D layered texture height

cudaDevAttrMaxTexture2DLayeredLayers

Maximum layers in a 2D layered texture

cudaDevAttrSurfaceAlignment

Alignment requirement for surfaces

cudaDevAttrConcurrentKernels

Device can possibly execute multiple kernels concurrently

cudaDevAttrEccEnabled

Device has ECC support enabled

cudaDevAttrPciBusId

PCI bus ID of the device

cudaDevAttrPciDeviceId

PCI device ID of the device

cudaDevAttrTccDriver

Device is using TCC driver model

cudaDevAttrMemoryClockRate

Peak memory clock frequency in kilohertz

cudaDevAttrGlobalMemoryBusWidth

Global memory bus width in bits

cudaDevAttrL2CacheSize

Size of L2 cache in bytes



cudaDevAttrMaxThreadsPerMultiProcessor
Maximum resident threads per multiprocessor

cudaDevAttrAsyncEngineCount
Number of asynchronous engines

cudaDevAttrUnifiedAddressing
Device shares a unified address space with the host

cudaDevAttrMaxTexture1DLayeredWidth
Maximum 1D layered texture width

cudaDevAttrMaxTexture1DLayeredLayers
Maximum layers in a 1D layered texture

cudaDevAttrMaxTexture2DGatherWidth
Maximum 2D texture width if *cudaArrayTextureGather* is set

cudaDevAttrMaxTexture2DGatherHeight
Maximum 2D texture height if *cudaArrayTextureGather* is set

cudaDevAttrMaxTexture3DWidthAlt
Alternate maximum 3D texture width

cudaDevAttrMaxTexture3DHeightAlt
Alternate maximum 3D texture height

cudaDevAttrMaxTexture3DDepthAlt
Alternate maximum 3D texture depth

cudaDevAttrPciDomainId
PCI domain ID of the device

cudaDevAttrTexturePitchAlignment
Pitch alignment requirement for textures

cudaDevAttrMaxTextureCubemapWidth
Maximum cubemap texture width/height

cudaDevAttrMaxTextureCubemapLayeredWidth
Maximum cubemap layered texture width/height

cudaDevAttrMaxTextureCubemapLayeredLayers
Maximum layers in a cubemap layered texture

cudaDevAttrMaxSurface1DWidth
Maximum 1D surface width

cudaDevAttrMaxSurface2DWidth
Maximum 2D surface width

cudaDevAttrMaxSurface2DHeight
Maximum 2D surface height

cudaDevAttrMaxSurface3DWidth
Maximum 3D surface width

cudaDevAttrMaxSurface3DHeight
Maximum 3D surface height

cudaDevAttrMaxSurface3DDepth
Maximum 3D surface depth

cudaDevAttrMaxSurface1DLayeredWidth
Maximum 1D layered surface width

cudaDevAttrMaxSurface1DLayeredLayers
Maximum layers in a 1D layered surface

cudaDevAttrMaxSurface2DLayeredWidth
Maximum 2D layered surface width



cudaDevAttrMaxSurface2DLayeredHeight
Maximum 2D layered surface height

cudaDevAttrMaxSurface2DLayeredLayers
Maximum layers in a 2D layered surface

cudaDevAttrMaxSurfaceCubemapWidth
Maximum cubemap surface width

cudaDevAttrMaxSurfaceCubemapLayeredWidth
Maximum cubemap layered surface width

cudaDevAttrMaxSurfaceCubemapLayeredLayers
Maximum layers in a cubemap layered surface

cudaDevAttrMaxTexture1DLinearWidth
Maximum 1D linear texture width

cudaDevAttrMaxTexture2DLinearWidth
Maximum 2D linear texture width

cudaDevAttrMaxTexture2DLinearHeight
Maximum 2D linear texture height

cudaDevAttrMaxTexture2DLinearPitch
Maximum 2D linear texture pitch in bytes

cudaDevAttrMaxTexture2DMipmappedWidth
Maximum mipmapped 2D texture width

cudaDevAttrMaxTexture2DMipmappedHeight
Maximum mipmapped 2D texture height

cudaDevAttrComputeCapabilityMajor
Major compute capability version number

cudaDevAttrComputeCapabilityMinor
Minor compute capability version number

cudaDevAttrMaxTexture1DMipmappedWidth
Maximum mipmapped 1D texture width

cudaDevAttrStreamPrioritiesSupported
Device supports stream priorities

cudaDevAttrGlobalL1CacheSupported
Device supports caching globals in L1

cudaDevAttrLocalL1CacheSupported
Device supports caching locals in L1

cudaDevAttrMaxSharedMemoryPerMultiprocessor
Maximum shared memory available per multiprocessor in bytes

cudaDevAttrMaxRegistersPerMultiprocessor
Maximum number of 32-bit registers available per multiprocessor

cudaDevAttrManagedMemory
Device can allocate managed memory on this system

cudaDevAttrIsMultiGpuBoard
Device is on a multi-GPU board

cudaDevAttrMultiGpuBoardGroupID
Unique identifier for a group of devices on the same multi-GPU board

enum cudaError

CUDA error types

Enumerator:

cudaSuccess

The API call returned with no errors. In the case of query calls, this can also mean that the operation being queried is complete (see **cudaEventQuery()** and **cudaStreamQuery()**).

cudaErrorMissingConfiguration

The device function being invoked (usually via **cudaLaunch()**) was not previously configured via the **cudaConfigureCall()** function.

cudaErrorMemoryAllocation

The API call failed because it was unable to allocate enough memory to perform the requested operation.

cudaErrorInitializationError

The API call failed because the CUDA driver and runtime could not be initialized.

cudaErrorLaunchFailure

An exception occurred on the device while executing a kernel. Common causes include dereferencing an invalid device pointer and accessing out of bounds shared memory. The device cannot be used until **cudaThreadExit()** is called. All existing device memory allocations are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorPriorLaunchFailure

This indicated that a previous kernel launch failed. This was previously used for device emulation of kernel launches.

Deprecated

This error return is deprecated as of CUDA 3.1. Device emulation mode was removed with the CUDA 3.1 release.

cudaErrorLaunchTimeout

This indicates that the device kernel took too long to execute. This can only occur if timeouts are enabled - see the device property **kernelExecTimeoutEnabled** for more information. The device cannot be used until **cudaThreadExit()** is called. All existing device memory allocations are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorLaunchOutOfResources

This indicates that a launch did not occur because it did not have appropriate resources. Although this error is similar to **cudaErrorInvalidConfiguration**, this error usually indicates that the user has attempted to pass too many arguments to the device kernel, or the kernel launch specifies too many threads for the kernel's register count.

cudaErrorInvalidDeviceFunction

The requested device function does not exist or is not compiled for the proper device architecture.

cudaErrorInvalidConfiguration

This indicates that a kernel launch is requesting resources that can never be satisfied by the current device. Requesting more shared memory per block than the device supports will trigger this error, as will requesting too many threads or blocks. See **cudaDeviceProp** for more device limitations.

cudaErrorInvalidDevice

This indicates that the device ordinal supplied by the user does not correspond to a valid CUDA device.

cudaErrorInvalidValue

This indicates that one or more of the parameters passed to the API call is not within an acceptable range of values.

cudaErrorInvalidPitchValue

This indicates that one or more of the pitch-related parameters passed to the API call is not within the acceptable range for pitch.

cudaErrorInvalidSymbol

This indicates that the symbol name/identifier passed to the API call is not a valid name or identifier.



cudaErrorMapBufferObjectFailed

This indicates that the buffer object could not be mapped.

cudaErrorUnmapBufferObjectFailed

This indicates that the buffer object could not be unmapped.

cudaErrorInvalidHostPointer

This indicates that at least one host pointer passed to the API call is not a valid host pointer.

cudaErrorInvalidDevicePointer

This indicates that at least one device pointer passed to the API call is not a valid device pointer.

cudaErrorInvalidTexture

This indicates that the texture passed to the API call is not a valid texture.

cudaErrorInvalidTextureBinding

This indicates that the texture binding is not valid. This occurs if you call **cudaGetTextureAlignmentOffset()** with an unbound texture.

cudaErrorInvalidChannelDescriptor

This indicates that the channel descriptor passed to the API call is not valid. This occurs if the format is not one of the formats specified by **cudaChannelFormatKind**, or if one of the dimensions is invalid.

cudaErrorInvalidMemcpyDirection

This indicates that the direction of the memcpy passed to the API call is not one of the types specified by **cudaMemcpyKind**.

cudaErrorAddressOfConstant

This indicated that the user has taken the address of a constant variable, which was forbidden up until the CUDA 3.1 release.

Deprecated

This error return is deprecated as of CUDA 3.1. Variables in constant memory may now have their address taken by the runtime via **cudaGetSymbolAddress()**.

cudaErrorTextureFetchFailed

This indicated that a texture fetch was not able to be performed. This was previously used for device emulation of texture operations.

Deprecated

This error return is deprecated as of CUDA 3.1. Device emulation mode was removed with the CUDA 3.1 release.

cudaErrorTextureNotBound

This indicated that a texture was not bound for access. This was previously used for device emulation of texture operations.

Deprecated

This error return is deprecated as of CUDA 3.1. Device emulation mode was removed with the CUDA 3.1 release.

cudaErrorSynchronizationError

This indicated that a synchronization operation had failed. This was previously used for some device emulation functions.

Deprecated

This error return is deprecated as of CUDA 3.1. Device emulation mode was removed with the CUDA 3.1 release.

cudaErrorInvalidFilterSetting

This indicates that a non-float texture was being accessed with linear filtering. This is not supported by CUDA.

cudaErrorInvalidNormSetting

This indicates that an attempt was made to read a non-float texture as a normalized float. This is not supported by CUDA.



cudaErrorMixedDeviceExecution

Mixing of device and device emulation code was not allowed.

Deprecated

This error return is deprecated as of CUDA 3.1. Device emulation mode was removed with the CUDA 3.1 release.

cudaErrorCudartUnloading

This indicates that a CUDA Runtime API call cannot be executed because it is being called during process shut down, at a point in time after CUDA driver has been unloaded.

cudaErrorUnknown

This indicates that an unknown internal error has occurred.

cudaErrorNotYetImplemented

This indicates that the API call is not yet implemented. Production releases of CUDA will never return this error.

Deprecated

This error return is deprecated as of CUDA 4.1.

cudaErrorMemoryValueTooLarge

This indicated that an emulated device pointer exceeded the 32-bit address range.

Deprecated

This error return is deprecated as of CUDA 3.1. Device emulation mode was removed with the CUDA 3.1 release.

cudaErrorInvalidResourceHandle

This indicates that a resource handle passed to the API call was not valid. Resource handles are opaque types like **cudaStream_t** and **cudaEvent_t**.

cudaErrorNotReady

This indicates that asynchronous operations issued previously have not completed yet. This result is not actually an error, but must be indicated differently than **cudaSuccess** (which indicates completion). Calls that may return this value include **cudaEventQuery()** and **cudaStreamQuery()**.

cudaErrorInsufficientDriver

This indicates that the installed NVIDIA CUDA driver is older than the CUDA runtime library. This is not a supported configuration. Users should install an updated NVIDIA display driver to allow the application to run.

cudaErrorSetOnActiveProcess

This indicates that the user has called **cudaSetValidDevices()**, **cudaSetDeviceFlags()**, **cudaD3D9SetDirect3DDevice()**, **cudaD3D10SetDirect3DDevice()**, **cudaD3D11SetDirect3DDevice()**, or **cudaVDPAUSetVDPAUDevice()** after initializing the CUDA runtime by calling non-device management operations (allocating memory and launching kernels are examples of non-device management operations). This error can also be returned if using runtime/driver interoperability and there is an existing CUcontext active on the host thread.

cudaErrorInvalidSurface

This indicates that the surface passed to the API call is not a valid surface.

cudaErrorNoDevice

This indicates that no CUDA-capable devices were detected by the installed CUDA driver.

cudaErrorECCUncorrectable

This indicates that an uncorrectable ECC error was detected during execution.

cudaErrorSharedObjectSymbolNotFound

This indicates that a link to a shared object failed to resolve.

cudaErrorSharedObjectInitFailed

This indicates that initialization of a shared object failed.



cudaErrorUnsupportedLimit

This indicates that the **cudaLimit** passed to the API call is not supported by the active device.

cudaErrorDuplicateVariableName

This indicates that multiple global or constant variables (across separate CUDA source files in the application) share the same string name.

cudaErrorDuplicateTextureName

This indicates that multiple textures (across separate CUDA source files in the application) share the same string name.

cudaErrorDuplicateSurfaceName

This indicates that multiple surfaces (across separate CUDA source files in the application) share the same string name.

cudaErrorDevicesUnavailable

This indicates that all CUDA devices are busy or unavailable at the current time. Devices are often busy/unavailable due to use of **cudaComputeModeExclusive**, **cudaComputeModeProhibited** or when long running CUDA kernels have filled up the GPU and are blocking new work from starting. They can also be unavailable due to memory constraints on a device that already has active CUDA work being performed.

cudaErrorInvalidKernelImage

This indicates that the device kernel image is invalid.

cudaErrorNoKernelImageForDevice

This indicates that there is no kernel image available that is suitable for the device. This can occur when a user specifies code generation options for a particular CUDA source file that do not include the corresponding device configuration.

cudaErrorIncompatibleDriverContext

This indicates that the current context is not compatible with this the CUDA Runtime. This can only occur if you are using CUDA Runtime/Driver interoperability and have created an existing Driver context using the driver API. The Driver context may be incompatible either because the Driver context was created using an older version of the API, because the Runtime API call expects a primary driver context and the Driver context is not primary, or because the Driver context has been destroyed. Please see **Interactions** with the CUDA Driver API' for more information.

cudaErrorPeerAccessAlreadyEnabled

This error indicates that a call to **cudaDeviceEnablePeerAccess()** is trying to re-enable peer addressing on from a context which has already had peer addressing enabled.

cudaErrorPeerAccessNotEnabled

This error indicates that **cudaDeviceDisablePeerAccess()** is trying to disable peer addressing which has not been enabled yet via **cudaDeviceEnablePeerAccess()**.

cudaErrorDeviceAlreadyInUse

This indicates that a call tried to access an exclusive-thread device that is already in use by a different thread.

cudaErrorProfilerDisabled

This indicates profiler is not initialized for this run. This can happen when the application is running with external profiling tools like visual profiler.

*cudaErrorProfilerNotInitialized***Deprecated**

This error return is deprecated as of CUDA 5.0. It is no longer an error to attempt to enable/disable the profiling via **cudaProfilerStart** or **cudaProfilerStop** without initialization.

*cudaErrorProfilerAlreadyStarted***Deprecated**

This error return is deprecated as of CUDA 5.0. It is no longer an error to call **cudaProfilerStart()** when profiling is already enabled.



*cudaErrorProfilerAlreadyStopped***Deprecated**

This error return is deprecated as of CUDA 5.0. It is no longer an error to call **cudaProfilerStop()** when profiling is already disabled.

cudaErrorAssert

An assert triggered in device code during kernel execution. The device cannot be used again until **cudaThreadExit()** is called. All existing allocations are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorTooManyPeers

This error indicates that the hardware resources required to enable peer access have been exhausted for one or more of the devices passed to **cudaEnablePeerAccess()**.

cudaErrorHostMemoryAlreadyRegistered

This error indicates that the memory range passed to **cudaHostRegister()** has already been registered.

cudaErrorHostMemoryNotRegistered

This error indicates that the pointer passed to **cudaHostUnregister()** does not correspond to any currently registered memory region.

cudaErrorOperatingSystem

This error indicates that an OS call failed.

cudaErrorPeerAccessUnsupported

This error indicates that P2P access is not supported across the given devices.

cudaErrorLaunchMaxDepthExceeded

This error indicates that a device runtime grid launch did not occur because the depth of the child grid would exceed the maximum supported number of nested grid launches.

cudaErrorLaunchFileScopedTex

This error indicates that a grid launch did not occur because the kernel uses file-scoped textures which are unsupported by the device runtime. Kernels launched via the device runtime only support textures created with the Texture Object API's.

cudaErrorLaunchFileScopedSurf

This error indicates that a grid launch did not occur because the kernel uses file-scoped surfaces which are unsupported by the device runtime. Kernels launched via the device runtime only support surfaces created with the Surface Object API's.

cudaErrorSyncDepthExceeded

This error indicates that a call to **cudaDeviceSynchronize** made from the device runtime failed because the call was made at grid depth greater than either the default (2 levels of grids) or user specified device limit **cudaLimitDevRuntimeSyncDepth**. To be able to synchronize on launched grids at a greater depth successfully, the maximum nested depth at which **cudaDeviceSynchronize** will be called must be specified with the **cudaLimitDevRuntimeSyncDepth** limit to the **cudaDeviceSetLimit** api before the host-side launch of a kernel using the device runtime. Keep in mind that additional levels of sync depth require the runtime to reserve large amounts of device memory that cannot be used for user allocations.

cudaErrorLaunchPendingCountExceeded

This error indicates that a device runtime grid launch failed because the launch would exceed the limit **cudaLimitDevRuntimePendingLaunchCount**. For this launch to proceed successfully, **cudaDeviceSetLimit** must be called to set the **cudaLimitDevRuntimePendingLaunchCount** to be higher than the upper bound of outstanding launches that can be issued to the device runtime. Keep in mind that raising the limit of pending device runtime launches will require the runtime to reserve device memory that cannot be used for user allocations.

cudaErrorNotPermitted

This error indicates the attempted operation is not permitted.



cudaErrorNotSupported

This error indicates the attempted operation is not supported on the current system or device.

cudaErrorHardwareStackError

Device encountered an error in the call stack during kernel execution, possibly due to stack corruption or exceeding the stack size limit. The context cannot be used, so it must be destroyed (and a new one should be created). All existing device memory allocations from this context are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorIllegalInstruction

The device encountered an illegal instruction during kernel execution. The context cannot be used, so it must be destroyed (and a new one should be created). All existing device memory allocations from this context are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorMisalignedAddress

The device encountered a load or store instruction on a memory address which is not aligned. The context cannot be used, so it must be destroyed (and a new one should be created). All existing device memory allocations from this context are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorInvalidAddressSpace

While executing a kernel, the device encountered an instruction which can only operate on memory locations in certain address spaces (global, shared, or local), but was supplied a memory address not belonging to an allowed address space. The context cannot be used, so it must be destroyed (and a new one should be created). All existing device memory allocations from this context are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorInvalidPc

The device encountered an invalid program counter. The context cannot be used, so it must be destroyed (and a new one should be created). All existing device memory allocations from this context are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorIllegalAddress

The device encountered a load or store instruction on an invalid memory address. The context cannot be used, so it must be destroyed (and a new one should be created). All existing device memory allocations from this context are invalid and must be reconstructed if the program is to continue using CUDA.

cudaErrorStartupFailure

This indicates an internal startup failure in the CUDA runtime.

cudaErrorApiFailureBase

Any unhandled CUDA driver error is added to this value and returned via the runtime. Production releases of CUDA should not return such errors.

Deprecated

This error return is deprecated as of CUDA 4.1.

enum cudaFuncCache

CUDA function cache configurations

Enumerator:*cudaFuncCachePreferNone*

Default function cache configuration, no preference

cudaFuncCachePreferShared

Prefer larger shared memory and smaller L1 cache

cudaFuncCachePreferL1

Prefer larger L1 cache and smaller shared memory

cudaFuncCachePreferEqual

Prefer equal size L1 cache and shared memory



enum cudaGraphicsCubeFace

CUDA graphics interop array indices for cube maps

Enumerator:*cudaGraphicsCubeFacePositiveX*

Positive X face of cubemap

cudaGraphicsCubeFaceNegativeX

Negative X face of cubemap

cudaGraphicsCubeFacePositiveY

Positive Y face of cubemap

cudaGraphicsCubeFaceNegativeY

Negative Y face of cubemap

cudaGraphicsCubeFacePositiveZ

Positive Z face of cubemap

cudaGraphicsCubeFaceNegativeZ

Negative Z face of cubemap

enum cudaGraphicsMapFlags

CUDA graphics interop map flags

Enumerator:*cudaGraphicsMapFlagsNone*

Default; Assume resource can be read/written

cudaGraphicsMapFlagsReadOnly

CUDA will not write to this resource

cudaGraphicsMapFlagsWriteDiscard

CUDA will only write to and will not read from this resource

enum cudaGraphicsRegisterFlags

CUDA graphics interop register flags

Enumerator:*cudaGraphicsRegisterFlagsNone*

Default

cudaGraphicsRegisterFlagsReadOnly

CUDA will not write to this resource

cudaGraphicsRegisterFlagsWriteDiscard

CUDA will only write to and will not read from this resource

cudaGraphicsRegisterFlagsSurfaceLoadStore

CUDA will bind this resource to a surface reference

cudaGraphicsRegisterFlagsTextureGather

CUDA will perform texture gather operations on this resource

enum cudaLimit

CUDA Limits

Enumerator:*cudaLimitStackSize*

GPU thread stack size

cudaLimitPrintfFifoSize

GPU printf/fprintf FIFO size

cudaLimitMallocHeapSize

GPU malloc heap size

cudaLimitDevRuntimeSyncDepth

GPU device runtime synchronize depth



cudaLimitDevRuntimePendingLaunchCount
GPU device runtime pending launch count

enum cudaMemcpyKind

CUDA memory copy types

Enumerator:

cudaMemcpyHostToHost
Host -> Host

cudaMemcpyHostToDevice
Host -> Device

cudaMemcpyDeviceToHost
Device -> Host

cudaMemcpyDeviceToDevice
Device -> Device

cudaMemcpyDefault
Default based unified virtual address space

enum cudaMemoryType

CUDA memory types

Enumerator:

cudaMemoryTypeHost
Host memory

cudaMemoryTypeDevice
Device memory

enum cudaOutputMode

CUDA Profiler Output modes

Enumerator:

cudaKeyValuePair
Output mode Key-Value pair format.

cudaCSV
Output mode Comma separated values format.

enum cudaResourceType

CUDA resource types

Enumerator:

cudaResourceTypeArray
Array resource

cudaResourceTypeMipmappedArray
Mipmapped array resource

cudaResourceTypeLinear
Linear resource

cudaResourceTypePitch2D
Pitch 2D resource

enum cudaResourceViewFormat

CUDA texture resource view formats

Enumerator:

cudaResViewFormatNone
No resource view format (use underlying resource format)

cudaResViewFormatUnsignedChar1
1 channel unsigned 8-bit integers



cudaResViewFormatUnsignedChar2
 2 channel unsigned 8-bit integers
cudaResViewFormatUnsignedChar4
 4 channel unsigned 8-bit integers
cudaResViewFormatSignedChar1
 1 channel signed 8-bit integers
cudaResViewFormatSignedChar2
 2 channel signed 8-bit integers
cudaResViewFormatSignedChar4
 4 channel signed 8-bit integers
cudaResViewFormatUnsignedShort1
 1 channel unsigned 16-bit integers
cudaResViewFormatUnsignedShort2
 2 channel unsigned 16-bit integers
cudaResViewFormatUnsignedShort4
 4 channel unsigned 16-bit integers
cudaResViewFormatSignedShort1
 1 channel signed 16-bit integers
cudaResViewFormatSignedShort2
 2 channel signed 16-bit integers
cudaResViewFormatSignedShort4
 4 channel signed 16-bit integers
cudaResViewFormatUnsignedInt1
 1 channel unsigned 32-bit integers
cudaResViewFormatUnsignedInt2
 2 channel unsigned 32-bit integers
cudaResViewFormatUnsignedInt4
 4 channel unsigned 32-bit integers
cudaResViewFormatSignedInt1
 1 channel signed 32-bit integers
cudaResViewFormatSignedInt2
 2 channel signed 32-bit integers
cudaResViewFormatSignedInt4
 4 channel signed 32-bit integers
cudaResViewFormatHalf1
 1 channel 16-bit floating point
cudaResViewFormatHalf2
 2 channel 16-bit floating point
cudaResViewFormatHalf4
 4 channel 16-bit floating point
cudaResViewFormatFloat1
 1 channel 32-bit floating point
cudaResViewFormatFloat2
 2 channel 32-bit floating point
cudaResViewFormatFloat4
 4 channel 32-bit floating point
cudaResViewFormatUnsignedBlockCompressed1
 Block compressed 1



cudaResViewFormatUnsignedBlockCompressed2
Block compressed 2

cudaResViewFormatUnsignedBlockCompressed3
Block compressed 3

cudaResViewFormatUnsignedBlockCompressed4
Block compressed 4 unsigned

cudaResViewFormatSignedBlockCompressed4
Block compressed 4 signed

cudaResViewFormatUnsignedBlockCompressed5
Block compressed 5 unsigned

cudaResViewFormatSignedBlockCompressed5
Block compressed 5 signed

cudaResViewFormatUnsignedBlockCompressed6H
Block compressed 6 unsigned half-float

cudaResViewFormatSignedBlockCompressed6H
Block compressed 6 signed half-float

cudaResViewFormatUnsignedBlockCompressed7
Block compressed 7

enum cudaSharedMemConfig

CUDA shared memory configuration

enum cudaSurfaceBoundaryMode

CUDA Surface boundary modes

Enumerator:

cudaBoundaryModeZero
Zero boundary mode

cudaBoundaryModeClamp
Clamp boundary mode

cudaBoundaryModeTrap
Trap boundary mode

enum cudaSurfaceFormatMode

CUDA Surface format modes

Enumerator:

cudaFormatModeForced
Forced format mode

cudaFormatModeAuto
Auto format mode

enum cudaTextureAddressMode

CUDA texture address modes

Enumerator:

cudaAddressModeWrap
Wrapping address mode

cudaAddressModeClamp
Clamp to edge address mode

cudaAddressModeMirror
Mirror address mode

cudaAddressModeBorder
Border address mode



enum cudaTextureFilterMode

CUDA texture filter modes

Enumerator:*cudaFilterModePoint*

Point filter mode

cudaFilterModeLinear

Linear filter mode

enum cudaTextureReadMode

CUDA texture read modes

Enumerator:*cudaReadModeElementType*

Read texture as specified element type

cudaReadModeNormalizedFloat

Read texture as normalized float

Author

Generated automatically by Doxygen from the source code.

