NAME

Data types used by CUDA driver −

Data Structures

struct **CUDA_ARRAY3D_DESCRIPTOR**
struct **CUDA_ARRAY_DESCRIPTOR**
struct **CUDA_MEMCPY2D**
struct **CUDA_MEMCPY3D**
struct **CUDA_MEMCPY3D_PEER**
struct **CUDA_POINTER_ATTRIBUTE_P2P_TOKENS**
struct **CUDA_RESOURCE_DESC**
struct **CUDA_RESOURCE_VIEW_DESC**
struct **CUDA_TEXTURE_DESC**
struct **CUdevprop**
struct **CUipcEventHandle**
struct **CUipcMemHandle**

Defines

#define **CU_IPC_HANDLE_SIZE** 64
#define **CU_LAUNCH_PARAM_BUFFER_POINTER** ((void*)0x01)
#define **CU_LAUNCH_PARAM_BUFFER_SIZE** ((void*)0x02)
#define **CU_LAUNCH_PARAM_END** ((void*)0x00)
#define **CU_MEMHOSTALLOC_DEVICEMAP** 0x02
#define **CU_MEMHOSTALLOC_PORTABLE** 0x01
#define **CU_MEMHOSTALLOC_WRITECOMBINED** 0x04
#define **CU_MEMHOSTREGISTER_DEVICEMAP** 0x02
#define **CU_MEMHOSTREGISTER_PORTABLE** 0x01
#define **CU_PARAM_TR_DEFAULT** -1
#define **CU_TRSA_OVERRIDE_FORMAT** 0x01
#define **CU_TRSF_NORMALIZED_COORDINATES** 0x02
#define **CU_TRSF_READ_AS_INTEGER** 0x01
#define **CU_TRSF_SRGB** 0x10
#define **CUDA_ARRAY3D_2DARRAY** 0x01
#define **CUDA_ARRAY3D_CUBEMAP** 0x04
#define **CUDA_ARRAY3D_DEPTH_TEXTURE** 0x10
#define **CUDA_ARRAY3D_LAYERED** 0x01
#define **CUDA_ARRAY3D_SURFACE_LDST** 0x02
#define **CUDA_ARRAY3D_TEXTURE_GATHER** 0x08
#define **CUDA_VERSION** 6050

Typedefs

typedef struct CUarray_st * **CUarray**
typedef struct CUctx_st * **CUcontext**
typedef int **CUdevice**
typedef unsigned int **CUdeviceptr**
typedef struct CUevent_st * **CUevent**
typedef struct CUfunc_st * **CUfunction**
typedef struct CUgraphicsResource_st * **CUgraphicsResource**
typedef struct CUmipmappedArray_st * **CUmipmappedArray**
typedef struct CUmod_st * **CUmodule**
typedef size_t(CUDA_CB * **CUoccupancyB2DSize**)(int blockSize)
typedef struct CUstream_st * **CUstream**
typedef void(CUDA_CB * **CUstreamCallback**)(**CUstream** hStream, **CUresult** status, void *userData)
typedef unsigned long long **CUsurfObject**
typedef struct CUsurfref_st * **CUsurfref**
typedef unsigned long long **CUtexObject**
typedef struct CUtexref_st * **CUtexref**



Enumerations

enum **CUaddress_mode** { **CU_TR_ADDRESS_MODE_WRAP** = 0,
CU_TR_ADDRESS_MODE_CLAMP = 1, **CU_TR_ADDRESS_MODE_MIRROR** = 2,
CU_TR_ADDRESS_MODE_BORDER = 3 }

enum **CUarray_cubemap_face** { **CU_CUBEMAP_FACE_POSITIVE_X** = 0x00,
CU_CUBEMAP_FACE_NEGATIVE_X = 0x01, **CU_CUBEMAP_FACE_POSITIVE_Y** = 0x02, **CU_CUBEMAP_FACE_NEGATIVE_Y** = 0x03,
CU_CUBEMAP_FACE_POSITIVE_Z = 0x04, **CU_CUBEMAP_FACE_NEGATIVE_Z** = 0x05 }

enum **CUarray_format** { **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 }

enum **CUcomputemode** { **CU_COMPUTEMODE_DEFAULT** = 0,
CU_COMPUTEMODE_EXCLUSIVE = 1, **CU_COMPUTEMODE_PROHIBITED** = 2,
CU_COMPUTEMODE_EXCLUSIVE_PROCESS = 3 }

enum **CUctx_flags** { **CU_CTX_SCHED_AUTO** = 0x00, **CU_CTX_SCHED_SPIN** = 0x01,
CU_CTX_SCHED_YIELD = 0x02, **CU_CTX_SCHED_BLOCKING_SYNC** = 0x04,
CU_CTX_BLOCKING_SYNC = 0x04, **CU_CTX_MAP_HOST** = 0x08,
CU_CTX_LMEM_RESIZE_TO_MAX = 0x10 }

enum **CUdevice_attribute** { **CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_BLOCK** = 1,
CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X = 2,
CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y = 3,
CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z = 4,
CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_X = 5,
CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y = 6,
CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z = 7,
CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK = 8,
CU_DEVICE_ATTRIBUTE_SHARED_MEMORY_PER_BLOCK = 8,
CU_DEVICE_ATTRIBUTE_TOTAL_CONSTANT_MEMORY = 9,
CU_DEVICE_ATTRIBUTE_WARP_SIZE = 10,
CU_DEVICE_ATTRIBUTE_MAX_PITCH = 11,
CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK = 12,
CU_DEVICE_ATTRIBUTE_REGISTERS_PER_BLOCK = 12,
CU_DEVICE_ATTRIBUTE_CLOCK_RATE = 13,
CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT = 14,
CU_DEVICE_ATTRIBUTE_GPU_OVERLAP = 15,
CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT = 16,
CU_DEVICE_ATTRIBUTE_KERNEL_EXEC_TIMEOUT = 17,
CU_DEVICE_ATTRIBUTE_INTEGRATED = 18,
CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST_MEMORY = 19,
CU_DEVICE_ATTRIBUTE_COMPUTE_MODE = 20,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_WIDTH = 21,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_WIDTH = 22,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_HEIGHT = 23,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_WIDTH = 24,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_HEIGHT = 25,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_DEPTH = 26,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_WIDTH = 27,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_HEIGHT = 28,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_LAYERS = 29,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_WIDTH = 27,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_HEIGHT = 28,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_NUMSLICES = 29,
CU_DEVICE_ATTRIBUTE_SURFACE_ALIGNMENT = 30,
CU_DEVICE_ATTRIBUTE_CONCURRENT_KERNELS = 31,
CU_DEVICE_ATTRIBUTE_ECC_ENABLED = 32,
CU_DEVICE_ATTRIBUTE_PCI_BUS_ID = 33,



CU_DEVICE_ATTRIBUTE_PCI_DEVICE_ID = 34,
CU_DEVICE_ATTRIBUTE_TCC_DRIVER = 35,
CU_DEVICE_ATTRIBUTE_MEMORY_CLOCK_RATE = 36,
CU_DEVICE_ATTRIBUTE_GLOBAL_MEMORY_BUS_WIDTH = 37,
CU_DEVICE_ATTRIBUTE_L2_CACHE_SIZE = 38,
CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_MULTIPROCESSOR = 39,
CU_DEVICE_ATTRIBUTE_ASYNC_ENGINE_COUNT = 40,
CU_DEVICE_ATTRIBUTE_UNIFIED_ADDRESSING = 41,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_LAYERED_WIDTH = 42,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_LAYERED_LAYERS = 43,
CU_DEVICE_ATTRIBUTE_CAN_TEX2D_GATHER = 44,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_GATHER_WIDTH = 45,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_GATHER_HEIGHT = 46,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_WIDTH_ALTERNATE = 47,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_HEIGHT_ALTERNATE = 48,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_DEPTH_ALTERNATE = 49,
CU_DEVICE_ATTRIBUTE_PCI_DOMAIN_ID = 50,
CU_DEVICE_ATTRIBUTE_TEXTURE_PITCH_ALIGNMENT = 51,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURECUBEMAP_WIDTH = 52,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURECUBEMAP_LAYERED_WIDTH = 53,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURECUBEMAP_LAYERED_LAYERS = 54, **CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE1D_WIDTH** = 55,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_WIDTH = 56,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_HEIGHT = 57,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE3D_WIDTH = 58,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE3D_HEIGHT = 59,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE3D_DEPTH = 60,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE1D_LAYERED_WIDTH = 61,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE1D_LAYERED_LAYERS = 62,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_LAYERED_WIDTH = 63,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_LAYERED_HEIGHT = 64,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_LAYERED_LAYERS = 65,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACECUBEMAP_WIDTH = 66,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACECUBEMAP_LAYERED_WIDTH = 67,
CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACECUBEMAP_LAYERED_LAYERS = 68, **CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_LINEAR_WIDTH** = 69,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LINEAR_WIDTH = 70,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LINEAR_HEIGHT = 71,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LINEAR_PITCH = 72,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_MIPMAPPED_WIDTH = 73,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_MIPMAPPED_HEIGHT = 74,
CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MAJOR = 75,
CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MINOR = 76,
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_MIPMAPPED_WIDTH = 77,
CU_DEVICE_ATTRIBUTE_STREAM_PRIORITIES_SUPPORTED = 78,
CU_DEVICE_ATTRIBUTE_GLOBAL_L1_CACHE_SUPPORTED = 79,
CU_DEVICE_ATTRIBUTE_LOCAL_L1_CACHE_SUPPORTED = 80,
CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_MULTIPROCESSOR = 81, **CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_MULTIPROCESSOR** = 82,
CU_DEVICE_ATTRIBUTE_MANAGED_MEMORY = 83,
CU_DEVICE_ATTRIBUTE_MULTI_GPU_BOARD = 84,
CU_DEVICE_ATTRIBUTE_MULTI_GPU_BOARD_GROUP_ID = 85 }

enum **CUevent_flags** { **CU_EVENT_DEFAULT** = 0x0, **CU_EVENT_BLOCKING_SYNC** = 0x1,
CU_EVENT_DISABLE_TIMING = 0x2, **CU_EVENT_INTERPROCESS** = 0x4 }

enum **CUfilter_mode** { **CU_TR_FILTER_MODE_POINT** = 0,
CU_TR_FILTER_MODE_LINEAR = 1 }

enum **CUfunc_cache** { **CU_FUNC_CACHE_PREFER_NONE** = 0x00,



CU_FUNC_CACHE_PREFER_SHARED = 0x01, **CU_FUNC_CACHE_PREFER_L1** = 0x02, **CU_FUNC_CACHE_PREFER_EQUAL** = 0x03 }

enum **CUfunction_attribute** { **CU_FUNC_ATTRIBUTE_MAX_THREADS_PER_BLOCK** = 0, **CU_FUNC_ATTRIBUTE_SHARED_SIZE_BYTES** = 1, **CU_FUNC_ATTRIBUTE_CONST_SIZE_BYTES** = 2, **CU_FUNC_ATTRIBUTE_LOCAL_SIZE_BYTES** = 3, **CU_FUNC_ATTRIBUTE_NUM_REGS** = 4, **CU_FUNC_ATTRIBUTE_PTX_VERSION** = 5, **CU_FUNC_ATTRIBUTE_BINARY_VERSION** = 6, **CU_FUNC_ATTRIBUTE_CACHE_MODE_CA** = 7 }

enum **CUgraphicsMapResourceFlags**

enum **CUgraphicsRegisterFlags**

enum **CUipcMem_flags** { **CU_IPC_MEM_LAZY_ENABLE_PEER_ACCESS** = 0x1 }

enum **CUjit_cacheMode** { **CU_JIT_CACHE_OPTION_NONE** = 0, **CU_JIT_CACHE_OPTION_CG**, **CU_JIT_CACHE_OPTION_CA** }

enum **CUjit_fallback** { **CU_PREFER_PTX** = 0, **CU_PREFER_BINARY** }

enum **CUjit_option** { **CU_JIT_MAX_REGISTERS** = 0, **CU_JIT_THREADS_PER_BLOCK**, **CU_JIT_WALL_TIME**, **CU_JIT_INFO_LOG_BUFFER**, **CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES**, **CU_JIT_ERROR_LOG_BUFFER**, **CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES**, **CU_JIT_OPTIMIZATION_LEVEL**, **CU_JIT_TARGET_FROM_CUCONTEXT**, **CU_JIT_TARGET**, **CU_JIT_FALLBACK_STRATEGY**, **CU_JIT_GENERATE_DEBUG_INFO**, **CU_JIT_LOG_VERBOSE**, **CU_JIT_GENERATE_LINE_INFO**, **CU_JIT_CACHE_MODE** }

enum **CUjit_target** { **CU_TARGET_COMPUTE_10** = 10, **CU_TARGET_COMPUTE_11** = 11, **CU_TARGET_COMPUTE_12** = 12, **CU_TARGET_COMPUTE_13** = 13, **CU_TARGET_COMPUTE_20** = 20, **CU_TARGET_COMPUTE_21** = 21, **CU_TARGET_COMPUTE_30** = 30, **CU_TARGET_COMPUTE_32** = 32, **CU_TARGET_COMPUTE_35** = 35, **CU_TARGET_COMPUTE_37** = 37, **CU_TARGET_COMPUTE_50** = 50 }

enum **CUjitInputType** { **CU_JIT_INPUT_CUBIN** = 0, **CU_JIT_INPUT_PTX**, **CU_JIT_INPUT_FATBINARY**, **CU_JIT_INPUT_OBJECT**, **CU_JIT_INPUT_LIBRARY** }

enum **CUlimit** { **CU_LIMIT_STACK_SIZE** = 0x00, **CU_LIMIT_PRINTF_FIFO_SIZE** = 0x01, **CU_LIMIT_MALLOC_HEAP_SIZE** = 0x02, **CU_LIMIT_DEV_RUNTIME_SYNC_DEPTH** = 0x03, **CU_LIMIT_DEV_RUNTIME_PENDING_LAUNCH_COUNT** = 0x04 }

enum **CUmemAttach_flags** { **CU_MEM_ATTACH_GLOBAL** = 0x1, **CU_MEM_ATTACH_HOST** = 0x2, **CU_MEM_ATTACH_SINGLE** = 0x4 }

enum **CUmemorytype** { **CU_MEMORYTYPE_HOST** = 0x01, **CU_MEMORYTYPE_DEVICE** = 0x02, **CU_MEMORYTYPE_ARRAY** = 0x03, **CU_MEMORYTYPE_UNIFIED** = 0x04 }

enum **CUpointer_attribute** { **CU_POINTER_ATTRIBUTE_CONTEXT** = 1, **CU_POINTER_ATTRIBUTE_MEMORY_TYPE** = 2, **CU_POINTER_ATTRIBUTE_DEVICE_POINTER** = 3, **CU_POINTER_ATTRIBUTE_HOST_POINTER** = 4, **CU_POINTER_ATTRIBUTE_P2P_TOKENS** = 5, **CU_POINTER_ATTRIBUTE_SYNC_MEMOPS** = 6, **CU_POINTER_ATTRIBUTE_BUFFER_ID** = 7, **CU_POINTER_ATTRIBUTE_IS_MANAGED** = 8 }

enum **CUresourcetype** { **CU_RESOURCE_TYPE_ARRAY** = 0x00, **CU_RESOURCE_TYPE_MIPMAPPED_ARRAY** = 0x01, **CU_RESOURCE_TYPE_LINEAR** = 0x02, **CU_RESOURCE_TYPE_PITCH2D** = 0x03 }

enum **CUresourceViewFormat** { **CU_RES_VIEW_FORMAT_NONE** = 0x00, **CU_RES_VIEW_FORMAT_UINT_1X8** = 0x01, **CU_RES_VIEW_FORMAT_UINT_2X8** = 0x02, **CU_RES_VIEW_FORMAT_UINT_4X8** = 0x03, **CU_RES_VIEW_FORMAT_SINT_1X8** = 0x04, **CU_RES_VIEW_FORMAT_SINT_2X8** = 0x05, **CU_RES_VIEW_FORMAT_SINT_4X8** = 0x06, **CU_RES_VIEW_FORMAT_UINT_1X16** = 0x07, **CU_RES_VIEW_FORMAT_UINT_2X16** = 0x08, **CU_RES_VIEW_FORMAT_UINT_4X16** = 0x09,



CU_RES_VIEW_FORMAT_SINT_1X16 = 0x0a,
CU_RES_VIEW_FORMAT_SINT_2X16 = 0x0b,
CU_RES_VIEW_FORMAT_SINT_4X16 = 0x0c,
CU_RES_VIEW_FORMAT_UINT_1X32 = 0x0d,
CU_RES_VIEW_FORMAT_UINT_2X32 = 0x0e,
CU_RES_VIEW_FORMAT_UINT_4X32 = 0x0f,
CU_RES_VIEW_FORMAT_SINT_1X32 = 0x10,
CU_RES_VIEW_FORMAT_SINT_2X32 = 0x11,
CU_RES_VIEW_FORMAT_SINT_4X32 = 0x12,
CU_RES_VIEW_FORMAT_FLOAT_1X16 = 0x13,
CU_RES_VIEW_FORMAT_FLOAT_2X16 = 0x14,
CU_RES_VIEW_FORMAT_FLOAT_4X16 = 0x15,
CU_RES_VIEW_FORMAT_FLOAT_1X32 = 0x16,
CU_RES_VIEW_FORMAT_FLOAT_2X32 = 0x17,
CU_RES_VIEW_FORMAT_FLOAT_4X32 = 0x18,
CU_RES_VIEW_FORMAT_UNSIGNED_BC1 = 0x19,
CU_RES_VIEW_FORMAT_UNSIGNED_BC2 = 0x1a,
CU_RES_VIEW_FORMAT_UNSIGNED_BC3 = 0x1b,
CU_RES_VIEW_FORMAT_UNSIGNED_BC4 = 0x1c,
CU_RES_VIEW_FORMAT_SIGNED_BC4 = 0x1d,
CU_RES_VIEW_FORMAT_UNSIGNED_BC5 = 0x1e,
CU_RES_VIEW_FORMAT_SIGNED_BC5 = 0x1f,
CU_RES_VIEW_FORMAT_UNSIGNED_BC6H = 0x20,
CU_RES_VIEW_FORMAT_SIGNED_BC6H = 0x21,
CU_RES_VIEW_FORMAT_UNSIGNED_BC7 = 0x22 }

enum **CUresult** { **CUDA_SUCCESS** = 0, **CUDA_ERROR_INVALID_VALUE** = 1,
CUDA_ERROR_OUT_OF_MEMORY = 2, **CUDA_ERROR_NOT_INITIALIZED** = 3,
CUDA_ERROR_DEINITIALIZED = 4, **CUDA_ERROR_PROFILER_DISABLED** = 5,
CUDA_ERROR_PROFILER_NOT_INITIALIZED = 6,
CUDA_ERROR_PROFILER_ALREADY_STARTED = 7,
CUDA_ERROR_PROFILER_ALREADY_STOPPED = 8,
CUDA_ERROR_NO_DEVICE = 100, **CUDA_ERROR_INVALID_DEVICE** = 101,
CUDA_ERROR_INVALID_IMAGE = 200, **CUDA_ERROR_INVALID_CONTEXT** = 201, **CUDA_ERROR_CONTEXT_ALREADY_CURRENT** = 202,
CUDA_ERROR_MAP_FAILED = 205, **CUDA_ERROR_UNMAP_FAILED** = 206,
CUDA_ERROR_ARRAY_IS_MAPPED = 207, **CUDA_ERROR_ALREADY_MAPPED** = 208, **CUDA_ERROR_NO_BINARY_FOR_GPU** = 209,
CUDA_ERROR_ALREADY_ACQUIRED = 210, **CUDA_ERROR_NOT_MAPPED** = 211, **CUDA_ERROR_NOT_MAPPED_AS_ARRAY** = 212,
CUDA_ERROR_NOT_MAPPED_AS_POINTER = 213,
CUDA_ERROR_ECC_UNCORRECTABLE = 214,
CUDA_ERROR_UNSUPPORTED_LIMIT = 215,
CUDA_ERROR_CONTEXT_ALREADY_IN_USE = 216,
CUDA_ERROR_PEER_ACCESS_UNSUPPORTED = 217,
CUDA_ERROR_INVALID_PTX = 218,
CUDA_ERROR_INVALID_GRAPHICS_CONTEXT = 219,
CUDA_ERROR_INVALID_SOURCE = 300, **CUDA_ERROR_FILE_NOT_FOUND** = 301, **CUDA_ERROR_SHARED_OBJECT_SYMBOL_NOT_FOUND** = 302,
CUDA_ERROR_SHARED_OBJECT_INIT_FAILED = 303,
CUDA_ERROR_OPERATING_SYSTEM = 304, **CUDA_ERROR_INVALID_HANDLE** = 400, **CUDA_ERROR_NOT_FOUND** = 500, **CUDA_ERROR_NOT_READY** = 600,
CUDA_ERROR_ILLEGAL_ADDRESS = 700,
CUDA_ERROR_LAUNCH_OUT_OF_RESOURCES = 701,
CUDA_ERROR_LAUNCH_TIMEOUT = 702,
CUDA_ERROR_LAUNCH_INCOMPATIBLE_TEXTURING = 703,
CUDA_ERROR_PEER_ACCESS_ALREADY_ENABLED = 704,
CUDA_ERROR_PEER_ACCESS_NOT_ENABLED = 705,
CUDA_ERROR_PRIMARY_CONTEXT_ACTIVE = 708,
CUDA_ERROR_CONTEXT_IS_DESTROYED = 709, **CUDA_ERROR_ASSERT** = 710,



CUDA_ERROR_TOO_MANY_PEERS = 711,
CUDA_ERROR_HOST_MEMORY_ALREADY_REGISTERED = 712,
CUDA_ERROR_HOST_MEMORY_NOT_REGISTERED = 713,
CUDA_ERROR_HARDWARE_STACK_ERROR = 714,
CUDA_ERROR_ILLEGAL_INSTRUCTION = 715,
CUDA_ERROR_MISALIGNED_ADDRESS = 716,
CUDA_ERROR_INVALID_ADDRESS_SPACE = 717, **CUDA_ERROR_INVALID_PC** = 718, **CUDA_ERROR_LAUNCH_FAILED** = 719, **CUDA_ERROR_NOT_PERMITTED** = 800, **CUDA_ERROR_NOT_SUPPORTED** = 801, **CUDA_ERROR_UNKNOWN** = 999 }

enum **CUsharedconfig** { **CU_SHARED_MEM_CONFIG_DEFAULT_BANK_SIZE** = 0x00,
CU_SHARED_MEM_CONFIG_FOUR_BYTE_BANK_SIZE = 0x01,
CU_SHARED_MEM_CONFIG_EIGHT_BYTE_BANK_SIZE = 0x02 }

enum **CUstream_flags** { **CU_STREAM_DEFAULT** = 0x0, **CU_STREAM_NON_BLOCKING** = 0x1 }

Define Documentation

#define CU_IPC_HANDLE_SIZE 64

CUDA IPC handle size

#define CU_LAUNCH_PARAM_BUFFER_POINTER ((void*)0x01)

Indicator that the next value in the `extra` parameter to **cuLaunchKernel** will be a pointer to a buffer containing all kernel parameters used for launching kernel `f`. This buffer needs to honor all alignment/padding requirements of the individual parameters. If **CU_LAUNCH_PARAM_BUFFER_SIZE** is not also specified in the `extra` array, then **CU_LAUNCH_PARAM_BUFFER_POINTER** will have no effect.

#define CU_LAUNCH_PARAM_BUFFER_SIZE ((void*)0x02)

Indicator that the next value in the `extra` parameter to **cuLaunchKernel** will be a pointer to a size_t which contains the size of the buffer specified with **CU_LAUNCH_PARAM_BUFFER_POINTER**. It is required that **CU_LAUNCH_PARAM_BUFFER_POINTER** also be specified in the `extra` array if the value associated with **CU_LAUNCH_PARAM_BUFFER_SIZE** is not zero.

#define CU_LAUNCH_PARAM_END ((void*)0x00)

End of array terminator for the `extra` parameter to **cuLaunchKernel**

#define CU_MEMHOSTALLOC_DEVICEMAP 0x02

If set, host memory is mapped into CUDA address space and **cuMemHostGetDevicePointer()** may be called on the host pointer. Flag for **cuMemHostAlloc()**

#define CU_MEMHOSTALLOC_PORTABLE 0x01

If set, host memory is portable between CUDA contexts. Flag for **cuMemHostAlloc()**

#define CU_MEMHOSTALLOC_WRITECOMBINED 0x04

If set, host memory is allocated as write-combined - fast to write, faster to DMA, slow to read except via SSE4 streaming load instruction (MOVNTDQA). Flag for **cuMemHostAlloc()**

#define CU_MEMHOSTREGISTER_DEVICEMAP 0x02

If set, host memory is mapped into CUDA address space and **cuMemHostGetDevicePointer()** may be called on the host pointer. Flag for **cuMemHostRegister()**

#define CU_MEMHOSTREGISTER_PORTABLE 0x01

If set, host memory is portable between CUDA contexts. Flag for **cuMemHostRegister()**

#define CU_PARAM_TR_DEFAULT -1

For texture references loaded into the module, use default texunit from texture reference.

#define CU_TRSA_OVERRIDE_FORMAT 0x01

Override the texref format with a format inferred from the array. Flag for **cuTexRefSetArray()**

#define CU_TRSF_NORMALIZED_COORDINATES 0x02

Use normalized texture coordinates in the range [0,1) instead of [0,dim). Flag for **cuTexRefSetFlags()**

#define CU_TRSF_READ_AS_INTEGER 0x01

Read the texture as integers rather than promoting the values to floats in the range [0,1]. Flag for **cuTexRefSetFlags()**



#define CU_TRSF_SRGB 0x10
Perform sRGB->linear conversion during texture read. Flag for **cuTexRefSetFlags()**

#define CUDA_ARRAY3D_2DARRAY 0x01
Deprecated, use CUDA_ARRAY3D_LAYERED

#define CUDA_ARRAY3D_CUBEMAP 0x04
If set, the CUDA array is a collection of six 2D arrays, representing faces of a cube. The width of such a CUDA array must be equal to its height, and Depth must be six. If **CUDA_ARRAY3D_LAYERED** flag is also set, then the CUDA array is a collection of cubemaps and Depth must be a multiple of six.

#define CUDA_ARRAY3D_DEPTH_TEXTURE 0x10
This flag if set indicates that the CUDA array is a DEPTH_TEXTURE.

#define CUDA_ARRAY3D_LAYERED 0x01
If set, the CUDA array is a collection of layers, where each layer is either a 1D or a 2D array and the Depth member of **CUDA_ARRAY3D_DESCRIPTOR** specifies the number of layers, not the depth of a 3D array.

#define CUDA_ARRAY3D_SURFACE_LDST 0x02
This flag must be set in order to bind a surface reference to the CUDA array

#define CUDA_ARRAY3D_TEXTURE_GATHER 0x08
This flag must be set in order to perform texture gather operations on a CUDA array.

#define CUDA_VERSION 6050
CUDA API version number

Typedef Documentation

typedef struct CUarray_st* CUarray
CUDA array

typedef struct CUctx_st* CUcontext
CUDA context

typedef int CUdevice
CUDA device

typedef unsigned int CUdeviceptr
CUDA device pointer

typedef struct CUevent_st* CUevent
CUDA event

typedef struct CUfunc_st* CUfunction
CUDA function

typedef struct CUgraphicsResource_st* CUgraphicsResource
CUDA graphics interop resource

typedef struct CUmipmappedArray_st* CUmipmappedArray
CUDA mipmapped array

typedef struct CUmod_st* CUmodule
CUDA module

typedef size_t(CUDA_CB * CUoccupancyB2DSize)(int blockSize)
Block size to per-block dynamic shared memory mapping for a certain kernel

Parameters:
blockSize Block size of the kernel.

Returns:
The dynamic shared memory needed by a block.

typedef struct CUstream_st* CUstream
CUDA stream

typedef void(CUDA_CB * CUstreamCallback)(CUstream hStream, CUresult status, void *userData)
CUDA stream callback

Parameters:



hStream The stream the callback was added to, as passed to **cuStreamAddCallback**. May be NULL.
status **CUDA_SUCCESS** or any persistent error on the stream.
userData User parameter provided at registration.

typedef unsigned long long CUsurfObject
CUDA surface object

typedef struct CUsurfref_st* CUsurfref
CUDA surface reference

typedef unsigned long long CUtexObject
CUDA texture object

typedef struct CUtexref_st* CUtexref
CUDA texture reference

Enumeration Type Documentation

enum CUaddress_mode
Texture reference addressing modes

Enumerator:

CU_TR_ADDRESS_MODE_WRAP
Wrapping address mode

CU_TR_ADDRESS_MODE_CLAMP
Clamp to edge address mode

CU_TR_ADDRESS_MODE_MIRROR
Mirror address mode

CU_TR_ADDRESS_MODE_BORDER
Border address mode

enum CUarray_cubemap_face
Array indices for cube faces

Enumerator:

CU_CUBEMAP_FACE_POSITIVE_X
Positive X face of cubemap

CU_CUBEMAP_FACE_NEGATIVE_X
Negative X face of cubemap

CU_CUBEMAP_FACE_POSITIVE_Y
Positive Y face of cubemap

CU_CUBEMAP_FACE_NEGATIVE_Y
Negative Y face of cubemap

CU_CUBEMAP_FACE_POSITIVE_Z
Positive Z face of cubemap

CU_CUBEMAP_FACE_NEGATIVE_Z
Negative Z face of cubemap

enum CUarray_format
Array formats

Enumerator:

CU_AD_FORMAT_UNSIGNED_INT8
Unsigned 8-bit integers

CU_AD_FORMAT_UNSIGNED_INT16
Unsigned 16-bit integers

CU_AD_FORMAT_UNSIGNED_INT32
Unsigned 32-bit integers



CU_AD_FORMAT_SIGNED_INT8
Signed 8-bit integers

CU_AD_FORMAT_SIGNED_INT16
Signed 16-bit integers

CU_AD_FORMAT_SIGNED_INT32
Signed 32-bit integers

CU_AD_FORMAT_HALF
16-bit floating point

CU_AD_FORMAT_FLOAT
32-bit floating point

enum CUcomputemode
Compute Modes

Enumerator:

CU_COMPUTEMODE_DEFAULT
Default compute mode (Multiple contexts allowed per device)

CU_COMPUTEMODE_EXCLUSIVE
Compute-exclusive-thread mode (Only one context used by a single thread can be present on this device at a time)

CU_COMPUTEMODE_PROHIBITED
Compute-prohibited mode (No contexts can be created on this device at this time)

CU_COMPUTEMODE_EXCLUSIVE_PROCESS
Compute-exclusive-process mode (Only one context used by a single process can be present on this device at a time)

enum CUctx_flags
Context creation flags

Enumerator:

CU_CTX_SCHED_AUTO
Automatic scheduling

CU_CTX_SCHED_SPIN
Set spin as default scheduling

CU_CTX_SCHED_YIELD
Set yield as default scheduling

CU_CTX_SCHED_BLOCKING_SYNC
Set blocking synchronization as default scheduling

CU_CTX_BLOCKING_SYNC
Set blocking synchronization as default scheduling

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

CU_CTX_MAP_HOST
Support mapped pinned allocations

CU_CTX_LMEM_RESIZE_TO_MAX
Keep local memory allocation after launch

enum CUdevice_attribute
Device properties

Enumerator:

CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_BLOCK
Maximum number of threads per block



CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X
Maximum block dimension X

CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y
Maximum block dimension Y

CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z
Maximum block dimension Z

CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_X
Maximum grid dimension X

CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y
Maximum grid dimension Y

CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z
Maximum grid dimension Z

CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK
Maximum shared memory available per block in bytes

CU_DEVICE_ATTRIBUTE_SHARED_MEMORY_PER_BLOCK
Deprecated, use CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK

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

CU_DEVICE_ATTRIBUTE_WARP_SIZE
Warp size in threads

CU_DEVICE_ATTRIBUTE_MAX_PITCH
Maximum pitch in bytes allowed by memory copies

CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK
Maximum number of 32-bit registers available per block

CU_DEVICE_ATTRIBUTE_REGISTERS_PER_BLOCK
Deprecated, use CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK

CU_DEVICE_ATTRIBUTE_CLOCK_RATE
Typical clock frequency in kilohertz

CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT
Alignment requirement for textures

CU_DEVICE_ATTRIBUTE_GPU_OVERLAP
Device can possibly copy memory and execute a kernel concurrently. Deprecated. Use instead CU_DEVICE_ATTRIBUTE_ASYNC_ENGINE_COUNT.

CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT
Number of multiprocessors on device

CU_DEVICE_ATTRIBUTE_KERNEL_EXEC_TIMEOUT
Specifies whether there is a run time limit on kernels

CU_DEVICE_ATTRIBUTE_INTEGRATED
Device is integrated with host memory

CU_DEVICE_ATTRIBUTE_CAN_MAP_HOST_MEMORY
Device can map host memory into CUDA address space

CU_DEVICE_ATTRIBUTE_COMPUTE_MODE
Compute mode (See **CUcomputemode** for details)

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_WIDTH
Maximum 1D texture width

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_WIDTH
Maximum 2D texture width



CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_HEIGHT
Maximum 2D texture height

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_WIDTH
Maximum 3D texture width

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_HEIGHT
Maximum 3D texture height

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_DEPTH
Maximum 3D texture depth

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_WIDTH
Maximum 2D layered texture width

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_HEIGHT
Maximum 2D layered texture height

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_LAYERS
Maximum layers in a 2D layered texture

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_WIDTH
Deprecated, use
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_WIDTH

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_HEIGHT
Deprecated, use
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_HEIGHT

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_ARRAY_NUMSLICES
Deprecated, use
CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LAYERED_LAYERS

CU_DEVICE_ATTRIBUTE_SURFACE_ALIGNMENT
Alignment requirement for surfaces

CU_DEVICE_ATTRIBUTE_CONCURRENT_KERNELS
Device can possibly execute multiple kernels concurrently

CU_DEVICE_ATTRIBUTE_ECC_ENABLED
Device has ECC support enabled

CU_DEVICE_ATTRIBUTE_PCI_BUS_ID
PCI bus ID of the device

CU_DEVICE_ATTRIBUTE_PCI_DEVICE_ID
PCI device ID of the device

CU_DEVICE_ATTRIBUTE_TCC_DRIVER
Device is using TCC driver model

CU_DEVICE_ATTRIBUTE_MEMORY_CLOCK_RATE
Peak memory clock frequency in kilohertz

CU_DEVICE_ATTRIBUTE_GLOBAL_MEMORY_BUS_WIDTH
Global memory bus width in bits

CU_DEVICE_ATTRIBUTE_L2_CACHE_SIZE
Size of L2 cache in bytes

CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_MULTIPROCESSOR
Maximum resident threads per multiprocessor

CU_DEVICE_ATTRIBUTE_ASYNC_ENGINE_COUNT
Number of asynchronous engines

CU_DEVICE_ATTRIBUTE_UNIFIED_ADDRESSING
Device shares a unified address space with the host

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_LAYERED_WIDTH
Maximum 1D layered texture width



CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_LAYERED_LAYERS
Maximum layers in a 1D layered texture

CU_DEVICE_ATTRIBUTE_CAN_TEX2D_GATHER
Deprecated, do not use.

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_GATHER_WIDTH
Maximum 2D texture width if CUDA_ARRAY3D_TEXTURE_GATHER is set

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_GATHER_HEIGHT
Maximum 2D texture height if CUDA_ARRAY3D_TEXTURE_GATHER is set

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_WIDTH_ALTERNATE
Alternate maximum 3D texture width

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_HEIGHT_ALTERNATE
Alternate maximum 3D texture height

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE3D_DEPTH_ALTERNATE
Alternate maximum 3D texture depth

CU_DEVICE_ATTRIBUTE_PCI_DOMAIN_ID
PCI domain ID of the device

CU_DEVICE_ATTRIBUTE_TEXTURE_PITCH_ALIGNMENT
Pitch alignment requirement for textures

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURECUBEMAP_WIDTH
Maximum cubemap texture width/height

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURECUBEMAP_LAYERED_WIDTH
Maximum cubemap layered texture width/height

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURECUBEMAP_LAYERED_LAYERS
Maximum layers in a cubemap layered texture

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE1D_WIDTH
Maximum 1D surface width

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_WIDTH
Maximum 2D surface width

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_HEIGHT
Maximum 2D surface height

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE3D_WIDTH
Maximum 3D surface width

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE3D_HEIGHT
Maximum 3D surface height

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE3D_DEPTH
Maximum 3D surface depth

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE1D_LAYERED_WIDTH
Maximum 1D layered surface width

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE1D_LAYERED_LAYERS
Maximum layers in a 1D layered surface

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_LAYERED_WIDTH
Maximum 2D layered surface width

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_LAYERED_HEIGHT
Maximum 2D layered surface height

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACE2D_LAYERED_LAYERS
Maximum layers in a 2D layered surface

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACECUBEMAP_WIDTH
Maximum cubemap surface width



CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACECUBEMAP_LAYERED_WIDTH
Maximum cubemap layered surface width

CU_DEVICE_ATTRIBUTE_MAXIMUM_SURFACECUBEMAP_LAYERED_LAYERS
Maximum layers in a cubemap layered surface

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_LINEAR_WIDTH
Maximum 1D linear texture width

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LINEAR_WIDTH
Maximum 2D linear texture width

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LINEAR_HEIGHT
Maximum 2D linear texture height

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_LINEAR_PITCH
Maximum 2D linear texture pitch in bytes

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_MIPMAPPED_WIDTH
Maximum mipmapped 2D texture width

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE2D_MIPMAPPED_HEIGHT
Maximum mipmapped 2D texture height

CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MAJOR
Major compute capability version number

CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MINOR
Minor compute capability version number

CU_DEVICE_ATTRIBUTE_MAXIMUM_TEXTURE1D_MIPMAPPED_WIDTH
Maximum mipmapped 1D texture width

CU_DEVICE_ATTRIBUTE_STREAM_PRIORITIES_SUPPORTED
Device supports stream priorities

CU_DEVICE_ATTRIBUTE_GLOBAL_L1_CACHE_SUPPORTED
Device supports caching globals in L1

CU_DEVICE_ATTRIBUTE_LOCAL_L1_CACHE_SUPPORTED
Device supports caching locals in L1

CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_MULTIPROCESSOR
Maximum shared memory available per multiprocessor in bytes

CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_MULTIPROCESSOR
Maximum number of 32-bit registers available per multiprocessor

CU_DEVICE_ATTRIBUTE_MANAGED_MEMORY
Device can allocate managed memory on this system

CU_DEVICE_ATTRIBUTE_MULTI_GPU_BOARD
Device is on a multi-GPU board

CU_DEVICE_ATTRIBUTE_MULTI_GPU_BOARD_GROUP_ID
Unique id for a group of devices on the same multi-GPU board

enum CUevent_flags
Event creation flags

Enumerator:

CU_EVENT_DEFAULT
Default event flag

CU_EVENT_BLOCKING_SYNC
Event uses blocking synchronization

CU_EVENT_DISABLE_TIMING
Event will not record timing data



CU_EVENT_INTERPROCESS
Event is suitable for interprocess use. CU_EVENT_DISABLE_TIMING must be set

enum CUfilter_mode
Texture reference filtering modes

Enumerator:

CU_TR_FILTER_MODE_POINT
Point filter mode

CU_TR_FILTER_MODE_LINEAR
Linear filter mode

enum CUfunc_cache
Function cache configurations

Enumerator:

CU_FUNC_CACHE_PREFER_NONE
no preference for shared memory or L1 (default)

CU_FUNC_CACHE_PREFER_SHARED
prefer larger shared memory and smaller L1 cache

CU_FUNC_CACHE_PREFER_L1
prefer larger L1 cache and smaller shared memory

CU_FUNC_CACHE_PREFER_EQUAL
prefer equal sized L1 cache and shared memory

enum CUfunction_attribute
Function properties

Enumerator:

CU_FUNC_ATTRIBUTE_MAX_THREADS_PER_BLOCK
The maximum number of threads per block, beyond which a launch of the function would fail. This number depends on both the function and the device on which the function is currently loaded.

CU_FUNC_ATTRIBUTE_SHARED_SIZE_BYTES
The size in bytes of statically-allocated shared memory required by this function. This does not include dynamically-allocated shared memory requested by the user at runtime.

CU_FUNC_ATTRIBUTE_CONST_SIZE_BYTES
The size in bytes of user-allocated constant memory required by this function.

CU_FUNC_ATTRIBUTE_LOCAL_SIZE_BYTES
The size in bytes of local memory used by each thread of this function.

CU_FUNC_ATTRIBUTE_NUM_REGS
The number of registers used by each thread of this function.

CU_FUNC_ATTRIBUTE_PTX_VERSION
The PTX virtual architecture version for which the function was compiled. This value is the major PTX version * 10 + the minor PTX version, so a PTX version 1.3 function would return the value 13. Note that this may return the undefined value of 0 for cubins compiled prior to CUDA 3.0.

CU_FUNC_ATTRIBUTE_BINARY_VERSION
The binary architecture version for which the function was compiled. This value is the major binary version * 10 + the minor binary version, so a binary version 1.3 function would return the value 13. Note that this will return a value of 10 for legacy cubins that do not have a properly-encoded binary architecture version.

CU_FUNC_ATTRIBUTE_CACHE_MODE_CA
The attribute to indicate whether the function has been compiled with user specified option '−Xptxas −−dlcm=ca' set .



enum CUgraphicsMapResourceFlags
Flags for mapping and unmapping interop resources

enum CUgraphicsRegisterFlags
Flags to register a graphics resource

enum CUipcMem_flags
CUDA Ipc Mem Flags

Enumerator:

CU_IPC_MEM_LAZY_ENABLE_PEER_ACCESS
Automatically enable peer access between remote devices as needed

enum CUjit_cacheMode
Caching modes for dlcm

Enumerator:

CU_JIT_CACHE_OPTION_NONE
Compile with no −dlcm flag specified

CU_JIT_CACHE_OPTION_CG
Compile with L1 cache disabled

CU_JIT_CACHE_OPTION_CA
Compile with L1 cache enabled

enum CUjit_fallback
Cubin matching fallback strategies

Enumerator:

CU_PREFER_PTX
Prefer to compile ptx if exact binary match not found

CU_PREFER_BINARY
Prefer to fall back to compatible binary code if exact match not found

enum CUjit_option
Online compiler and linker options

Enumerator:

CU_JIT_MAX_REGISTERS
Max number of registers that a thread may use.
Option type: unsigned int
Applies to: compiler only

CU_JIT_THREADS_PER_BLOCK
IN: Specifies minimum number of threads per block to target compilation for
OUT: Returns the number of threads the compiler actually targeted. This restricts the resource utilization fo the compiler (e.g. max registers) such that a block with the given number of threads should be able to launch based on register limitations. Note, this option does not currently take into account any other resource limitations, such as shared memory utilization.
Cannot be combined with **CU_JIT_TARGET**.
Option type: unsigned int
Applies to: compiler only

CU_JIT_WALL_TIME
Overwrites the option value with the total wall clock time, in milliseconds, spent in the compiler and linker
Option type: float
Applies to: compiler and linker

CU_JIT_INFO_LOG_BUFFER
Pointer to a buffer in which to print any log messages that are informational in nature (the buffer size is specified via option **CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES**)
Option type: char *
Applies to: compiler and linker



CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES
IN: Log buffer size in bytes. Log messages will be capped at this size (including null terminator)
OUT: Amount of log buffer filled with messages
Option type: unsigned int
Applies to: compiler and linker

CU_JIT_ERROR_LOG_BUFFER
Pointer to a buffer in which to print any log messages that reflect errors (the buffer size is specified via option **CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES**)
Option type: char *
Applies to: compiler and linker

CU_JIT_ERROR_LOG_BUFFER_SIZE_BYTES
IN: Log buffer size in bytes. Log messages will be capped at this size (including null terminator)
OUT: Amount of log buffer filled with messages
Option type: unsigned int
Applies to: compiler and linker

CU_JIT_OPTIMIZATION_LEVEL
Level of optimizations to apply to generated code (0 - 4), with 4 being the default and highest level of optimizations.
Option type: unsigned int
Applies to: compiler only

CU_JIT_TARGET_FROM_CUCONTEXT
No option value required. Determines the target based on the current attached context (default)
Option type: No option value needed
Applies to: compiler and linker

CU_JIT_TARGET
Target is chosen based on supplied **CUjit_target**. Cannot be combined with **CU_JIT_THREADS_PER_BLOCK**.
Option type: unsigned int for enumerated type **CUjit_target**
Applies to: compiler and linker

CU_JIT_FALLBACK_STRATEGY
Specifies choice of fallback strategy if matching cubin is not found. Choice is based on supplied **CUjit_fallback**. This option cannot be used with cuLink* APIs as the linker requires exact matches.
Option type: unsigned int for enumerated type **CUjit_fallback**
Applies to: compiler only

CU_JIT_GENERATE_DEBUG_INFO
Specifies whether to create debug information in output (−g) (0: false, default)
Option type: int
Applies to: compiler and linker

CU_JIT_LOG_VERBOSE
Generate verbose log messages (0: false, default)
Option type: int
Applies to: compiler and linker

CU_JIT_GENERATE_LINE_INFO
Generate line number information (−lineinfo) (0: false, default)
Option type: int
Applies to: compiler only

CU_JIT_CACHE_MODE
Specifies whether to enable caching explicitly (−dlcm)
Choice is based on supplied CUjit_cacheMode_enum.
Option type: unsigned int for enumerated type CUjit_cacheMode_enum
Applies to: compiler only



enum CUjit_target

Online compilation targets

Enumerator:

CU_TARGET_COMPUTE_10
: Compute device class 1.0

CU_TARGET_COMPUTE_11
: Compute device class 1.1

CU_TARGET_COMPUTE_12
: Compute device class 1.2

CU_TARGET_COMPUTE_13
: Compute device class 1.3

CU_TARGET_COMPUTE_20
: Compute device class 2.0

CU_TARGET_COMPUTE_21
: Compute device class 2.1

CU_TARGET_COMPUTE_30
: Compute device class 3.0

CU_TARGET_COMPUTE_32
: Compute device class 3.2

CU_TARGET_COMPUTE_35
: Compute device class 3.5

CU_TARGET_COMPUTE_37
: Compute device class 3.7

CU_TARGET_COMPUTE_50
: Compute device class 5.0

enum CUjitInputType

Device code formats

Enumerator:

CU_JIT_INPUT_CUBIN
: Compiled device-class-specific device code
Applicable options: none

CU_JIT_INPUT_PTX
: PTX source code
Applicable options: PTX compiler options

CU_JIT_INPUT_FATBINARY
: Bundle of multiple cubins and/or PTX of some device code
Applicable options: PTX compiler options, **CU_JIT_FALLBACK_STRATEGY**

CU_JIT_INPUT_OBJECT
: Host object with embedded device code
Applicable options: PTX compiler options, **CU_JIT_FALLBACK_STRATEGY**

CU_JIT_INPUT_LIBRARY
: Archive of host objects with embedded device code
Applicable options: PTX compiler options, **CU_JIT_FALLBACK_STRATEGY**

enum CUlimit

Limits

Enumerator:

CU_LIMIT_STACK_SIZE
: GPU thread stack size



CU_LIMIT_PRINTF_FIFO_SIZE
GPU printf FIFO size

CU_LIMIT_MALLOC_HEAP_SIZE
GPU malloc heap size

CU_LIMIT_DEV_RUNTIME_SYNC_DEPTH
GPU device runtime launch synchronize depth

CU_LIMIT_DEV_RUNTIME_PENDING_LAUNCH_COUNT
GPU device runtime pending launch count

enum CUmemAttach_flags
CUDA Mem Attach Flags

Enumerator:

CU_MEM_ATTACH_GLOBAL
Memory can be accessed by any stream on any device

CU_MEM_ATTACH_HOST
Memory cannot be accessed by any stream on any device

CU_MEM_ATTACH_SINGLE
Memory can only be accessed by a single stream on the associated device

enum CUmemorytype
Memory types

Enumerator:

CU_MEMORYTYPE_HOST
Host memory

CU_MEMORYTYPE_DEVICE
Device memory

CU_MEMORYTYPE_ARRAY
Array memory

CU_MEMORYTYPE_UNIFIED
Unified device or host memory

enum CUpointer_attribute
Pointer information

Enumerator:

CU_POINTER_ATTRIBUTE_CONTEXT
The **CUcontext** on which a pointer was allocated or registered

CU_POINTER_ATTRIBUTE_MEMORY_TYPE
The **CUmemorytype** describing the physical location of a pointer

CU_POINTER_ATTRIBUTE_DEVICE_POINTER
The address at which a pointer's memory may be accessed on the device

CU_POINTER_ATTRIBUTE_HOST_POINTER
The address at which a pointer's memory may be accessed on the host

CU_POINTER_ATTRIBUTE_P2P_TOKENS
A pair of tokens for use with the nv-p2p.h Linux kernel interface

CU_POINTER_ATTRIBUTE_SYNC_MEMOPS
Synchronize every synchronous memory operation initiated on this region

CU_POINTER_ATTRIBUTE_BUFFER_ID
A process-wide unique ID for an allocated memory region

CU_POINTER_ATTRIBUTE_IS_MANAGED
Indicates if the pointer points to managed memory



enum CUresourcetype

Resource types

Enumerator:

CU_RESOURCE_TYPE_ARRAY
: Array resoure

CU_RESOURCE_TYPE_MIPMAPPED_ARRAY
: Mipmapped array resource

CU_RESOURCE_TYPE_LINEAR
: Linear resource

CU_RESOURCE_TYPE_PITCH2D
: Pitch 2D resource

enum CUresourceViewFormat

Resource view format

Enumerator:

CU_RES_VIEW_FORMAT_NONE
: No resource view format (use underlying resource format)

CU_RES_VIEW_FORMAT_UINT_1X8
: 1 channel unsigned 8-bit integers

CU_RES_VIEW_FORMAT_UINT_2X8
: 2 channel unsigned 8-bit integers

CU_RES_VIEW_FORMAT_UINT_4X8
: 4 channel unsigned 8-bit integers

CU_RES_VIEW_FORMAT_SINT_1X8
: 1 channel signed 8-bit integers

CU_RES_VIEW_FORMAT_SINT_2X8
: 2 channel signed 8-bit integers

CU_RES_VIEW_FORMAT_SINT_4X8
: 4 channel signed 8-bit integers

CU_RES_VIEW_FORMAT_UINT_1X16
: 1 channel unsigned 16-bit integers

CU_RES_VIEW_FORMAT_UINT_2X16
: 2 channel unsigned 16-bit integers

CU_RES_VIEW_FORMAT_UINT_4X16
: 4 channel unsigned 16-bit integers

CU_RES_VIEW_FORMAT_SINT_1X16
: 1 channel signed 16-bit integers

CU_RES_VIEW_FORMAT_SINT_2X16
: 2 channel signed 16-bit integers

CU_RES_VIEW_FORMAT_SINT_4X16
: 4 channel signed 16-bit integers

CU_RES_VIEW_FORMAT_UINT_1X32
: 1 channel unsigned 32-bit integers

CU_RES_VIEW_FORMAT_UINT_2X32
: 2 channel unsigned 32-bit integers

CU_RES_VIEW_FORMAT_UINT_4X32
: 4 channel unsigned 32-bit integers

CU_RES_VIEW_FORMAT_SINT_1X32
: 1 channel signed 32-bit integers



CU_RES_VIEW_FORMAT_SINT_2X32
2 channel signed 32-bit integers

CU_RES_VIEW_FORMAT_SINT_4X32
4 channel signed 32-bit integers

CU_RES_VIEW_FORMAT_FLOAT_1X16
1 channel 16-bit floating point

CU_RES_VIEW_FORMAT_FLOAT_2X16
2 channel 16-bit floating point

CU_RES_VIEW_FORMAT_FLOAT_4X16
4 channel 16-bit floating point

CU_RES_VIEW_FORMAT_FLOAT_1X32
1 channel 32-bit floating point

CU_RES_VIEW_FORMAT_FLOAT_2X32
2 channel 32-bit floating point

CU_RES_VIEW_FORMAT_FLOAT_4X32
4 channel 32-bit floating point

CU_RES_VIEW_FORMAT_UNSIGNED_BC1
Block compressed 1

CU_RES_VIEW_FORMAT_UNSIGNED_BC2
Block compressed 2

CU_RES_VIEW_FORMAT_UNSIGNED_BC3
Block compressed 3

CU_RES_VIEW_FORMAT_UNSIGNED_BC4
Block compressed 4 unsigned

CU_RES_VIEW_FORMAT_SIGNED_BC4
Block compressed 4 signed

CU_RES_VIEW_FORMAT_UNSIGNED_BC5
Block compressed 5 unsigned

CU_RES_VIEW_FORMAT_SIGNED_BC5
Block compressed 5 signed

CU_RES_VIEW_FORMAT_UNSIGNED_BC6H
Block compressed 6 unsigned half-float

CU_RES_VIEW_FORMAT_SIGNED_BC6H
Block compressed 6 signed half-float

CU_RES_VIEW_FORMAT_UNSIGNED_BC7
Block compressed 7

enum CUresult
Error codes

Enumerator:

CUDA_SUCCESS
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 **cuEventQuery()** and **cuStreamQuery()**).

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

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



CUDA_ERROR_NOT_INITIALIZED
: This indicates that the CUDA driver has not been initialized with **cuInit()** or that initialization has failed.

CUDA_ERROR_DEINITIALIZED
: This indicates that the CUDA driver is in the process of shutting down.

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

CUDA_ERROR_PROFILER_NOT_INITIALIZED
: **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 **cuProfilerStart** or **cuProfilerStop** without initialization.

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

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

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

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

CUDA_ERROR_INVALID_IMAGE
: This indicates that the device kernel image is invalid. This can also indicate an invalid CUDA module.

CUDA_ERROR_INVALID_CONTEXT
: This most frequently indicates that there is no context bound to the current thread. This can also be returned if the context passed to an API call is not a valid handle (such as a context that has had **cuCtxDestroy()** invoked on it). This can also be returned if a user mixes different API versions (i.e. 3010 context with 3020 API calls). See **cuCtxGetApiVersion()** for more details.

CUDA_ERROR_CONTEXT_ALREADY_CURRENT
: This indicated that the context being supplied as a parameter to the API call was already the active context.

Deprecated
: This error return is deprecated as of CUDA 3.2. It is no longer an error to attempt to push the active context via **cuCtxPushCurrent()**.

CUDA_ERROR_MAP_FAILED
: This indicates that a map or register operation has failed.

CUDA_ERROR_UNMAP_FAILED
: This indicates that an unmap or unregister operation has failed.

CUDA_ERROR_ARRAY_IS_MAPPED
: This indicates that the specified array is currently mapped and thus cannot be destroyed.

CUDA_ERROR_ALREADY_MAPPED
: This indicates that the resource is already mapped.

CUDA_ERROR_NO_BINARY_FOR_GPU
: 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.



CUDA_ERROR_ALREADY_ACQUIRED
: This indicates that a resource has already been acquired.

CUDA_ERROR_NOT_MAPPED
: This indicates that a resource is not mapped.

CUDA_ERROR_NOT_MAPPED_AS_ARRAY
: This indicates that a mapped resource is not available for access as an array.

CUDA_ERROR_NOT_MAPPED_AS_POINTER
: This indicates that a mapped resource is not available for access as a pointer.

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

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

CUDA_ERROR_CONTEXT_ALREADY_IN_USE
: This indicates that the **CUcontext** passed to the API call can only be bound to a single CPU thread at a time but is already bound to a CPU thread.

CUDA_ERROR_PEER_ACCESS_UNSUPPORTED
: This indicates that peer access is not supported across the given devices.

CUDA_ERROR_INVALID_PTX
: This indicates that a PTX JIT compilation failed.

CUDA_ERROR_INVALID_GRAPHICS_CONTEXT
: This indicates an error with OpenGL or DirectX context.

CUDA_ERROR_INVALID_SOURCE
: This indicates that the device kernel source is invalid.

CUDA_ERROR_FILE_NOT_FOUND
: This indicates that the file specified was not found.

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

CUDA_ERROR_SHARED_OBJECT_INIT_FAILED
: This indicates that initialization of a shared object failed.

CUDA_ERROR_OPERATING_SYSTEM
: This indicates that an OS call failed.

CUDA_ERROR_INVALID_HANDLE
: This indicates that a resource handle passed to the API call was not valid. Resource handles are opaque types like **CUstream** and **CUevent**.

CUDA_ERROR_NOT_FOUND
: This indicates that a named symbol was not found. Examples of symbols are global/constant variable names, texture names, and surface names.

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

CUDA_ERROR_ILLEGAL_ADDRESS
: While executing a kernel, 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.

CUDA_ERROR_LAUNCH_OUT_OF_RESOURCES
: This indicates that a launch did not occur because it did not have appropriate resources. 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. Passing



arguments of the wrong size (i.e. a 64-bit pointer when a 32-bit int is expected) is equivalent to passing too many arguments and can also result in this error.

CUDA_ERROR_LAUNCH_TIMEOUT
: This indicates that the device kernel took too long to execute. This can only occur if timeouts are enabled - see the device attribute **CU_DEVICE_ATTRIBUTE_KERNEL_EXEC_TIMEOUT** for more information. The context cannot be used (and must be destroyed similar to **CUDA_ERROR_LAUNCH_FAILED**). All existing device memory allocations from this context are invalid and must be reconstructed if the program is to continue using CUDA.

CUDA_ERROR_LAUNCH_INCOMPATIBLE_TEXTURING
: This error indicates a kernel launch that uses an incompatible texturing mode.

CUDA_ERROR_PEER_ACCESS_ALREADY_ENABLED
: This error indicates that a call to **cuCtxEnablePeerAccess()** is trying to re-enable peer access to a context which has already had peer access to it enabled.

CUDA_ERROR_PEER_ACCESS_NOT_ENABLED
: This error indicates that **cuCtxDisablePeerAccess()** is trying to disable peer access which has not been enabled yet via **cuCtxEnablePeerAccess()**.

CUDA_ERROR_PRIMARY_CONTEXT_ACTIVE
: This error indicates that the primary context for the specified device has already been initialized.

CUDA_ERROR_CONTEXT_IS_DESTROYED
: This error indicates that the context current to the calling thread has been destroyed using **cuCtxDestroy**, or is a primary context which has not yet been initialized.

CUDA_ERROR_ASSERT
: A device-side assert triggered during kernel execution. The context cannot be used anymore, and must be destroyed. All existing device memory allocations from this context are invalid and must be reconstructed if the program is to continue using CUDA.

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

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

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

CUDA_ERROR_HARDWARE_STACK_ERROR
: While executing a kernel, the device encountered a stack error. This can be 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.

CUDA_ERROR_ILLEGAL_INSTRUCTION
: While executing a kernel, the device encountered an illegal instruction. 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.

CUDA_ERROR_MISALIGNED_ADDRESS
: While executing a kernel, 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.



CUDA_ERROR_INVALID_ADDRESS_SPACE
: 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.

CUDA_ERROR_INVALID_PC
: While executing a kernel, the device program counter wrapped its 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.

CUDA_ERROR_LAUNCH_FAILED
: 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 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.

CUDA_ERROR_NOT_PERMITTED
: This error indicates that the attempted operation is not permitted.

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

CUDA_ERROR_UNKNOWN
: This indicates that an unknown internal error has occurred.

enum CUsharedconfig
: Shared memory configurations

Enumerator:

CU_SHARED_MEM_CONFIG_DEFAULT_BANK_SIZE
: set default shared memory bank size

CU_SHARED_MEM_CONFIG_FOUR_BYTE_BANK_SIZE
: set shared memory bank width to four bytes

CU_SHARED_MEM_CONFIG_EIGHT_BYTE_BANK_SIZE
: set shared memory bank width to eight bytes

enum CUstream_flags
: Stream creation flags

Enumerator:

CU_STREAM_DEFAULT
: Default stream flag

CU_STREAM_NON_BLOCKING
: Stream does not synchronize with stream 0 (the NULL stream)

Author

Generated automatically by Doxygen from the source code.

