enum cudaError

CUDA error types

###### **Values**

cudaSuccess = 0

The API call returned with no errors. In the case of query calls, this also means that the operation being queried is complete (see [cudaEventQuery()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html#group__CUDART__EVENT_1g2bf738909b4a059023537eaa29d8a5b7) and [cudaStreamQuery()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__STREAM.html#group__CUDART__STREAM_1g2021adeb17905c7ec2a3c1bf125c5435)).

cudaErrorInvalidValue = 1

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

cudaErrorMemoryAllocation = 2

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

cudaErrorInitializationError = 3

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

cudaErrorCudartUnloading = 4

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.

cudaErrorProfilerDisabled = 5

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 = 6

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000048)

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](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PROFILER.html#group__CUDART__PROFILER_1gf536d75bb382356e10e3b4e89f4a5374) or [cudaProfilerStop](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PROFILER.html#group__CUDART__PROFILER_1g826922d9d1d0090d4a9a6b8b249cebb5) without initialization.

cudaErrorProfilerAlreadyStarted = 7

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000049)

This error return is deprecated as of CUDA 5.0. It is no longer an error to call [cudaProfilerStart()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PROFILER.html#group__CUDART__PROFILER_1gf536d75bb382356e10e3b4e89f4a5374) when profiling is already enabled.

cudaErrorProfilerAlreadyStopped = 8

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000050)

This error return is deprecated as of CUDA 5.0. It is no longer an error to call [cudaProfilerStop()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PROFILER.html#group__CUDART__PROFILER_1g826922d9d1d0090d4a9a6b8b249cebb5) when profiling is already disabled.

cudaErrorInvalidConfiguration = 9

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](https://docs.nvidia.com/cuda/cuda-runtime-api/structcudaDeviceProp.html#structcudaDeviceProp) for more device limitations.

cudaErrorInvalidPitchValue = 12

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 = 13

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

cudaErrorInvalidHostPointer = 16

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000051)

This error return is deprecated as of CUDA 10.1.

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

cudaErrorInvalidDevicePointer = 17

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000052)

This error return is deprecated as of CUDA 10.1.

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

cudaErrorInvalidTexture = 18

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

cudaErrorInvalidTextureBinding = 19

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

cudaErrorInvalidChannelDescriptor = 20

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](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1g8085eac5cb54b4228f3619a60f235119), or if one of the dimensions is invalid.

cudaErrorInvalidMemcpyDirection = 21

This indicates that the direction of the memcpy passed to the API call is not one of the types specified by [cudaMemcpyKind](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1g18fa99055ee694244a270e4d5101e95b).

cudaErrorAddressOfConstant = 22

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000053)

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()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html#group__CUDART__MEMORY_1g4f513be54d3794667c2017146b3d6a2b).

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

cudaErrorTextureFetchFailed = 23

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000054)

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

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

cudaErrorTextureNotBound = 24

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000055)

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

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

cudaErrorSynchronizationError = 25

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000056)

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

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

cudaErrorInvalidFilterSetting = 26

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

cudaErrorInvalidNormSetting = 27

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

cudaErrorMixedDeviceExecution = 28

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000057)

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

Mixing of device and device emulation code was not allowed.

cudaErrorNotYetImplemented = 31

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000058)

This error return is deprecated as of CUDA 4.1.

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

cudaErrorMemoryValueTooLarge = 32

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000059)

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

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

cudaErrorStubLibrary = 34

This indicates that the CUDA driver that the application has loaded is a stub library. Applications that run with the stub rather than a real driver loaded will result in CUDA API returning this error.

cudaErrorInsufficientDriver = 35

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.

cudaErrorCallRequiresNewerDriver = 36

This indicates that the API call requires a newer CUDA driver than the one currently installed. Users should install an updated NVIDIA CUDA driver to allow the API call to succeed.

cudaErrorInvalidSurface = 37

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

cudaErrorDuplicateVariableName = 43

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

cudaErrorDuplicateTextureName = 44

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

cudaErrorDuplicateSurfaceName = 45

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

cudaErrorDevicesUnavailable = 46

This indicates that all CUDA devices are busy or unavailable at the current time. Devices are often busy/unavailable due to use of [cudaComputeModeProhibited](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg7eb25f5413a962faad0956d92bae10d0fc71b88518e4501544d6e65b5f3671b6), [cudaComputeModeExclusiveProcess](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg7eb25f5413a962faad0956d92bae10d02cd032834fecbec513ea1018145b111d), 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.

cudaErrorIncompatibleDriverContext = 49

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](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DRIVER.html#group__CUDART__DRIVER) with the CUDA Driver API" for more information.

cudaErrorMissingConfiguration = 52

The device function being invoked (usually via [cudaLaunchKernel()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EXECUTION.html#group__CUDART__EXECUTION_1g5064cdf5d8e6741ace56fd8be951783c)) was not previously configured via the cudaConfigureCall() function.

cudaErrorPriorLaunchFailure = 53

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000060)

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

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

cudaErrorLaunchMaxDepthExceeded = 65

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 = 66

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 = 67

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 = 68

This error indicates that a call to [cudaDeviceSynchronize](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html#group__CUDART__DEVICE_1g10e20b05a95f638a4071a655503df25d) made from the device runtime failed because the call was made at grid depth greater than than either the default (2 levels of grids) or user specified device limit [cudaLimitDevRuntimeSyncDepth](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg4c4b34c054d383b0e9a63ab0ffc9365123c4900be4af436cb769e4c72d07be11). To be able to synchronize on launched grids at a greater depth successfully, the maximum nested depth at which [cudaDeviceSynchronize](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html#group__CUDART__DEVICE_1g10e20b05a95f638a4071a655503df25d) will be called must be specified with the [cudaLimitDevRuntimeSyncDepth](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg4c4b34c054d383b0e9a63ab0ffc9365123c4900be4af436cb769e4c72d07be11) limit to the [cudaDeviceSetLimit](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html#group__CUDART__DEVICE_1g05956f16eaa47ef3a4efee84563ccb7d) 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. Note that [cudaDeviceSynchronize](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html#group__CUDART__DEVICE_1g10e20b05a95f638a4071a655503df25d) made from device runtime is only supported on devices of compute capability < 9.0.

cudaErrorLaunchPendingCountExceeded = 69

This error indicates that a device runtime grid launch failed because the launch would exceed the limit [cudaLimitDevRuntimePendingLaunchCount](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg4c4b34c054d383b0e9a63ab0ffc9365118712cb05d2c3efaeea73afba823d916). For this launch to proceed successfully, [cudaDeviceSetLimit](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html#group__CUDART__DEVICE_1g05956f16eaa47ef3a4efee84563ccb7d) must be called to set the [cudaLimitDevRuntimePendingLaunchCount](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg4c4b34c054d383b0e9a63ab0ffc9365118712cb05d2c3efaeea73afba823d916) 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.

cudaErrorInvalidDeviceFunction = 98

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

cudaErrorNoDevice = 100

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

cudaErrorInvalidDevice = 101

This indicates that the device ordinal supplied by the user does not correspond to a valid CUDA device or that the action requested is invalid for the specified device.

cudaErrorDeviceNotLicensed = 102

This indicates that the device doesn't have a valid Grid License.

cudaErrorSoftwareValidityNotEstablished = 103

By default, the CUDA runtime may perform a minimal set of self-tests, as well as CUDA driver tests, to establish the validity of both. Introduced in CUDA 11.2, this error return indicates that at least one of these tests has failed and the validity of either the runtime or the driver could not be established.

cudaErrorStartupFailure = 127

This indicates an internal startup failure in the CUDA runtime.

cudaErrorInvalidKernelImage = 200

This indicates that the device kernel image is invalid.

cudaErrorDeviceUninitialized = 201

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()](https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__CTX.html#group__CUDA__CTX_1g27a365aebb0eb548166309f58a1e8b8e) 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()](https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__CTX.html#group__CUDA__CTX_1g088a90490dafca5893ef6fbebc8de8fb) for more details.

cudaErrorMapBufferObjectFailed = 205

This indicates that the buffer object could not be mapped.

cudaErrorUnmapBufferObjectFailed = 206

This indicates that the buffer object could not be unmapped.

cudaErrorArrayIsMapped = 207

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

cudaErrorAlreadyMapped = 208

This indicates that the resource is already mapped.

cudaErrorNoKernelImageForDevice = 209

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.

cudaErrorAlreadyAcquired = 210

This indicates that a resource has already been acquired.

cudaErrorNotMapped = 211

This indicates that a resource is not mapped.

cudaErrorNotMappedAsArray = 212

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

cudaErrorNotMappedAsPointer = 213

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

cudaErrorECCUncorrectable = 214

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

cudaErrorUnsupportedLimit = 215

This indicates that the [cudaLimit](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1g4c4b34c054d383b0e9a63ab0ffc93651) passed to the API call is not supported by the active device.

cudaErrorDeviceAlreadyInUse = 216

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

cudaErrorPeerAccessUnsupported = 217

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

cudaErrorInvalidPtx = 218

A PTX compilation failed. The runtime may fall back to compiling PTX if an application does not contain a suitable binary for the current device.

cudaErrorInvalidGraphicsContext = 219

This indicates an error with the OpenGL or DirectX context.

cudaErrorNvlinkUncorrectable = 220

This indicates that an uncorrectable NVLink error was detected during the execution.

cudaErrorJitCompilerNotFound = 221

This indicates that the PTX JIT compiler library was not found. The JIT Compiler library is used for PTX compilation. The runtime may fall back to compiling PTX if an application does not contain a suitable binary for the current device.

cudaErrorUnsupportedPtxVersion = 222

This indicates that the provided PTX was compiled with an unsupported toolchain. The most common reason for this, is the PTX was generated by a compiler newer than what is supported by the CUDA driver and PTX JIT compiler.

cudaErrorJitCompilationDisabled = 223

This indicates that the JIT compilation was disabled. The JIT compilation compiles PTX. The runtime may fall back to compiling PTX if an application does not contain a suitable binary for the current device.

cudaErrorUnsupportedExecAffinity = 224

This indicates that the provided execution affinity is not supported by the device.

cudaErrorInvalidSource = 300

This indicates that the device kernel source is invalid.

cudaErrorFileNotFound = 301

This indicates that the file specified was not found.

cudaErrorSharedObjectSymbolNotFound = 302

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

cudaErrorSharedObjectInitFailed = 303

This indicates that initialization of a shared object failed.

cudaErrorOperatingSystem = 304

This error indicates that an OS call failed.

cudaErrorInvalidResourceHandle = 400

This indicates that a resource handle passed to the API call was not valid. Resource handles are opaque types like [cudaStream\_t](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1ge15d9c8b7a240312b533d6122558085a) and [cudaEvent\_t](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gea2f543a9fc0e52fe4ae712920fd1247).

cudaErrorIllegalState = 401

This indicates that a resource required by the API call is not in a valid state to perform the requested operation.

cudaErrorSymbolNotFound = 500

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

cudaErrorNotReady = 600

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](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg3f51e3575c2178246db0a94a430e0038e355f04607d824883b4a50662830d591) (which indicates completion). Calls that may return this value include [cudaEventQuery()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html#group__CUDART__EVENT_1g2bf738909b4a059023537eaa29d8a5b7) and [cudaStreamQuery()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__STREAM.html#group__CUDART__STREAM_1g2021adeb17905c7ec2a3c1bf125c5435).

cudaErrorIllegalAddress = 700

The device encountered a load or store instruction on an invalid memory address. This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorLaunchOutOfResources = 701

This indicates that a launch did not occur because it did not have appropriate resources. Although this error is similar to [cudaErrorInvalidConfiguration](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg3f51e3575c2178246db0a94a430e0038d3934977826b2e1488dfeb7e0614e776), 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.

cudaErrorLaunchTimeout = 702

This indicates that the device kernel took too long to execute. This can only occur if timeouts are enabled - see the device property [kernelExecTimeoutEnabled](https://docs.nvidia.com/cuda/cuda-runtime-api/structcudaDeviceProp.html#structcudaDeviceProp_19a63114766c4d2309f00403c1bf056c8) for more information. This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorLaunchIncompatibleTexturing = 703

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

cudaErrorPeerAccessAlreadyEnabled = 704

This error indicates that a call to [cudaDeviceEnablePeerAccess()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PEER.html#group__CUDART__PEER_1g2b0adabf90db37e5cfddc92cbb2589f3) is trying to re-enable peer addressing on from a context which has already had peer addressing enabled.

cudaErrorPeerAccessNotEnabled = 705

This error indicates that [cudaDeviceDisablePeerAccess()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PEER.html#group__CUDART__PEER_1g9663734ad02653207ad6836053bf572e) is trying to disable peer addressing which has not been enabled yet via [cudaDeviceEnablePeerAccess()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PEER.html#group__CUDART__PEER_1g2b0adabf90db37e5cfddc92cbb2589f3).

cudaErrorSetOnActiveProcess = 708

This indicates that the user has called [cudaSetValidDevices()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html#group__CUDART__DEVICE_1g1b9336c70f2299405f67a4f8496d7cfe), [cudaSetDeviceFlags()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html#group__CUDART__DEVICE_1g69e73c7dda3fc05306ae7c811a690fac), [cudaD3D9SetDirect3DDevice()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__D3D9.html#group__CUDART__D3D9_1g0733a6f0bdc8aca6c6812da66c6d814a), [cudaD3D10SetDirect3DDevice](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__D3D10__DEPRECATED.html#group__CUDART__D3D10__DEPRECATED_1g7c942225222284fbfc3794a4ed6edf98), [cudaD3D11SetDirect3DDevice()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__D3D11__DEPRECATED.html#group__CUDART__D3D11__DEPRECATED_1g533524c773e7ca4ef57c0698b157d670), or [cudaVDPAUSetVDPAUDevice()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__VDPAU.html#group__CUDART__VDPAU_1g2a9ccfe8e6c87fe694bafd8cfcc67586) 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](https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__TYPES.html#group__CUDA__TYPES_1gf9f5bd81658f866613785b3a0bb7d7d9) active on the host thread.

cudaErrorContextIsDestroyed = 709

This error indicates that the context current to the calling thread has been destroyed using [cuCtxDestroy](https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__CTX.html#group__CUDA__CTX_1g27a365aebb0eb548166309f58a1e8b8e), or is a primary context which has not yet been initialized.

cudaErrorAssert = 710

An assert triggered in device code during kernel execution. The device cannot be used again. All existing allocations are invalid. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorTooManyPeers = 711

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 = 712

This error indicates that the memory range passed to [cudaHostRegister()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html#group__CUDART__MEMORY_1ge8d5c17670f16ac4fc8fcb4181cb490c) has already been registered.

cudaErrorHostMemoryNotRegistered = 713

This error indicates that the pointer passed to [cudaHostUnregister()](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html#group__CUDART__MEMORY_1g81fd4101862bbefdb42a62d60e515eea) does not correspond to any currently registered memory region.

cudaErrorHardwareStackError = 714

Device encountered an error in the call stack during kernel execution, possibly due to stack corruption or exceeding the stack size limit. This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorIllegalInstruction = 715

The device encountered an illegal instruction during kernel execution This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorMisalignedAddress = 716

The device encountered a load or store instruction on a memory address which is not aligned. This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorInvalidAddressSpace = 717

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. This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorInvalidPc = 718

The device encountered an invalid program counter. This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorLaunchFailure = 719

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. Less common cases can be system specific - more information about these cases can be found in the system specific user guide. This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorCooperativeLaunchTooLarge = 720

This error indicates that the number of blocks launched per grid for a kernel that was launched via either [cudaLaunchCooperativeKernel](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__HIGHLEVEL.html#group__CUDART__HIGHLEVEL_1gc0210b928f9bf4e212af07d35ac11d67) or [cudaLaunchCooperativeKernelMultiDevice](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EXECUTION.html#group__CUDART__EXECUTION_1g63685d849da7565b5774f5321a342f05) exceeds the maximum number of blocks as allowed by [cudaOccupancyMaxActiveBlocksPerMultiprocessor](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__HIGHLEVEL.html#group__CUDART__HIGHLEVEL_1g5a5d67a3c907371559ba692195e8a38c) or [cudaOccupancyMaxActiveBlocksPerMultiprocessorWithFlags](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__HIGHLEVEL.html#group__CUDART__HIGHLEVEL_1g603b86b20b37823253ff89fe8688ba83) times the number of multiprocessors as specified by the device attribute [cudaDevAttrMultiProcessorCount](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html#group__CUDART__TYPES_1gg49e2f8c2c0bd6fe264f2fc970912e5cd74de9ed281f917c8e4ef272da09f338a).

cudaErrorNotPermitted = 800

This error indicates the attempted operation is not permitted.

cudaErrorNotSupported = 801

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

cudaErrorSystemNotReady = 802

This error indicates that the system is not yet ready to start any CUDA work. To continue using CUDA, verify the system configuration is in a valid state and all required driver daemons are actively running. More information about this error can be found in the system specific user guide.

cudaErrorSystemDriverMismatch = 803

This error indicates that there is a mismatch between the versions of the display driver and the CUDA driver. Refer to the compatibility documentation for supported versions.

cudaErrorCompatNotSupportedOnDevice = 804

This error indicates that the system was upgraded to run with forward compatibility but the visible hardware detected by CUDA does not support this configuration. Refer to the compatibility documentation for the supported hardware matrix or ensure that only supported hardware is visible during initialization via the CUDA\_VISIBLE\_DEVICES environment variable.

cudaErrorMpsConnectionFailed = 805

This error indicates that the MPS client failed to connect to the MPS control daemon or the MPS server.

cudaErrorMpsRpcFailure = 806

This error indicates that the remote procedural call between the MPS server and the MPS client failed.

cudaErrorMpsServerNotReady = 807

This error indicates that the MPS server is not ready to accept new MPS client requests. This error can be returned when the MPS server is in the process of recovering from a fatal failure.

cudaErrorMpsMaxClientsReached = 808

This error indicates that the hardware resources required to create MPS client have been exhausted.

cudaErrorMpsMaxConnectionsReached = 809

This error indicates the the hardware resources required to device connections have been exhausted.

cudaErrorMpsClientTerminated = 810

This error indicates that the MPS client has been terminated by the server. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorCdpNotSupported = 811

This error indicates, that the program is using CUDA Dynamic Parallelism, but the current configuration, like MPS, does not support it.

cudaErrorCdpVersionMismatch = 812

This error indicates, that the program contains an unsupported interaction between different versions of CUDA Dynamic Parallelism.

cudaErrorStreamCaptureUnsupported = 900

The operation is not permitted when the stream is capturing.

cudaErrorStreamCaptureInvalidated = 901

The current capture sequence on the stream has been invalidated due to a previous error.

cudaErrorStreamCaptureMerge = 902

The operation would have resulted in a merge of two independent capture sequences.

cudaErrorStreamCaptureUnmatched = 903

The capture was not initiated in this stream.

cudaErrorStreamCaptureUnjoined = 904

The capture sequence contains a fork that was not joined to the primary stream.

cudaErrorStreamCaptureIsolation = 905

A dependency would have been created which crosses the capture sequence boundary. Only implicit in-stream ordering dependencies are allowed to cross the boundary.

cudaErrorStreamCaptureImplicit = 906

The operation would have resulted in a disallowed implicit dependency on a current capture sequence from cudaStreamLegacy.

cudaErrorCapturedEvent = 907

The operation is not permitted on an event which was last recorded in a capturing stream.

cudaErrorStreamCaptureWrongThread = 908

A stream capture sequence not initiated with the cudaStreamCaptureModeRelaxed argument to [cudaStreamBeginCapture](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__STREAM.html#group__CUDART__STREAM_1g793d7d4e474388ddfda531603dc34aa3) was passed to [cudaStreamEndCapture](https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__STREAM.html#group__CUDART__STREAM_1gf5a0efebc818054ceecd1e3e5e76d93e) in a different thread.

cudaErrorTimeout = 909

This indicates that the wait operation has timed out.

cudaErrorGraphExecUpdateFailure = 910

This error indicates that the graph update was not performed because it included changes which violated constraints specific to instantiated graph update.

cudaErrorExternalDevice = 911

This indicates that an async error has occurred in a device outside of CUDA. If CUDA was waiting for an external device's signal before consuming shared data, the external device signaled an error indicating that the data is not valid for consumption. This leaves the process in an inconsistent state and any further CUDA work will return the same error. To continue using CUDA, the process must be terminated and relaunched.

cudaErrorInvalidClusterSize = 912

This indicates that a kernel launch error has occurred due to cluster misconfiguration.

cudaErrorUnknown = 999

This indicates that an unknown internal error has occurred.

cudaErrorApiFailureBase = 10000

###### [**Deprecated**](https://docs.nvidia.com/cuda/cuda-runtime-api/deprecated.html#deprecated__deprecated_1_deprecated000061)

This error return is deprecated as of CUDA 4.1.

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