



# CUDA Debugger API

## API Reference Manual

# Table of Contents

<b>Chapter 1. Release Notes.....</b>	<b>1</b>
1.1. 11.8 Release.....	1
1.2. 12.3 Release.....	1
1.3. 7.0 Release.....	2
1.4. 6.5 Release.....	2
<b>Chapter 2. Introduction.....</b>	<b>3</b>
2.1. Debugger API.....	3
2.2. ELF and DWARF.....	4
2.3. ABI Support.....	5
2.4. Exception Reporting.....	6
2.5. Attaching and Detaching.....	6
<b>Chapter 3. Modules.....</b>	<b>8</b>
3.1. General.....	8
CUDBGResult.....	8
3.2. Initialization.....	10
CUDBGAPI_st::finalize.....	11
CUDBGAPI_st::getSupportedDebuggerCapabilities.....	11
CUDBGAPI_st::initialize.....	11
3.4. Breakpoints.....	12
CUDBGAPI_st::getAdjustedCodeAddress.....	12
CUDBGAPI_st::setBreakpoint.....	12
CUDBGAPI_st::setBreakpoint31.....	13
CUDBGAPI_st::unsetBreakpoint.....	13
CUDBGAPI_st::unsetBreakpoint31.....	14
3.5. Device State Inspection.....	14
CUDBGAPI_st::generateCoredump.....	14
CUDBGAPI_st::getConstBankAddress.....	15
CUDBGAPI_st::getConstBankAddress123.....	15
CUDBGAPI_st::getLoadedFunctionInfo.....	16
CUDBGAPI_st::getLoadedFunctionInfo118.....	17
CUDBGAPI_st::getManagedMemoryRegionInfo.....	17
CUDBGAPI_st::memcheckReadErrorAddress.....	18
CUDBGAPI_st::readActiveLanes.....	19
CUDBGAPI_st::readAllVirtualReturnAddresses.....	20
CUDBGAPI_st::readBlockIdx.....	21

CUDBGAPI_st::readBlockIdx32.....	21
CUDBGAPI_st::readBrokenWarps.....	22
CUDBGAPI_st::readCallDepth.....	23
CUDBGAPI_st::readCallDepth32.....	24
CUDBGAPI_st::readCCRegister.....	24
CUDBGAPI_st::readClusterIdx.....	25
CUDBGAPI_st::readCodeMemory.....	26
CUDBGAPI_st::readConstMemory.....	27
CUDBGAPI_st::readErrorPC.....	28
CUDBGAPI_st::readGenericMemory.....	28
CUDBGAPI_st::readGlobalMemory.....	29
CUDBGAPI_st::readGlobalMemory31.....	30
CUDBGAPI_st::readGlobalMemory55.....	31
CUDBGAPI_st::readGridId.....	32
CUDBGAPI_st::readGridId50.....	33
CUDBGAPI_st::readLaneException.....	34
CUDBGAPI_st::readLaneStatus.....	34
CUDBGAPI_st::readLocalMemory.....	35
CUDBGAPI_st::readParamMemory.....	36
CUDBGAPI_st::readPC.....	37
CUDBGAPI_st::readPinnedMemory.....	38
CUDBGAPI_st::readPredicates.....	39
CUDBGAPI_st::readRegister.....	40
CUDBGAPI_st::readRegisterRange.....	41
CUDBGAPI_st::readReturnAddress.....	42
CUDBGAPI_st::readReturnAddress32.....	43
CUDBGAPI_st::readSharedMemory.....	44
CUDBGAPI_st::readSmException.....	45
CUDBGAPI_st::readSyscallCallDepth.....	45
CUDBGAPI_st::readTextureMemory.....	46
CUDBGAPI_st::readTextureMemoryBindless.....	47
CUDBGAPI_st::readThreadId.....	48
CUDBGAPI_st::readUniformPredicates.....	49
CUDBGAPI_st::readUniformRegisterRange.....	50
CUDBGAPI_st::readValidLanes.....	50
CUDBGAPI_st::readValidWarps.....	51
CUDBGAPI_st::readVirtualPC.....	52
CUDBGAPI_st::readVirtualReturnAddress.....	53

CUDBGAPI_st::readVirtualReturnAddress32.....	54
CUDBGAPI_st::readWarpState.....	55
CUDBGAPI_st::readWarpState60.....	55
CUDBGAPI_st::writePinnedMemory.....	56
CUDBGAPI_st::writePredicates.....	57
CUDBGAPI_st::writeUniformPredicates.....	58
3.6. Device State Alteration.....	58
CUDBGAPI_st::writeCCRegister.....	59
CUDBGAPI_st::writeGenericMemory.....	60
CUDBGAPI_st::writeGlobalMemory.....	61
CUDBGAPI_st::writeGlobalMemory31.....	61
CUDBGAPI_st::writeGlobalMemory55.....	62
CUDBGAPI_st::writeLocalMemory.....	63
CUDBGAPI_st::writeParamMemory.....	64
CUDBGAPI_st::writeRegister.....	65
CUDBGAPI_st::writeSharedMemory.....	66
CUDBGAPI_st::writeUniformRegister.....	67
3.7. Grid Properties.....	67
CUDBGGridInfo.....	67
CUDBGGridStatus.....	67
CUDBGAPI_st::getBlockDim.....	68
CUDBGAPI_st::getClusterDim.....	69
CUDBGAPI_st::getElfImage.....	69
CUDBGAPI_st::getElfImage32.....	70
CUDBGAPI_st::getGridAttribute.....	71
CUDBGAPI_st::getGridAttributes.....	71
CUDBGAPI_st::getGridDim.....	72
CUDBGAPI_st::getGridDim32.....	72
CUDBGAPI_st::getGridInfo.....	73
CUDBGAPI_st::getGridInfo55.....	74
CUDBGAPI_st::getGridStatus.....	74
CUDBGAPI_st::getGridStatus50.....	75
CUDBGAPI_st::getTID.....	75
3.8. Device Properties.....	76
CUDBGAPI_st::getDeviceName.....	76
CUDBGAPI_st::getDeviceType.....	76
CUDBGAPI_st::getNumDevices.....	77
CUDBGAPI_st::getNumLanes.....	77

CUDBGAPI_st::getNumPredicates.....	78
CUDBGAPI_st::getNumRegisters.....	79
CUDBGAPI_st::getNumSMs.....	79
CUDBGAPI_st::getNumUniformPredicates.....	80
CUDBGAPI_st::getNumUniformRegisters.....	80
CUDBGAPI_st::getNumWarps.....	81
CUDBGAPI_st::getSmType.....	81
3.9. DWARF Utilities.....	82
CUDBGAPI_st::disassemble.....	82
CUDBGAPI_st::getElfImageByHandle.....	83
CUDBGAPI_st::getHostAddrFromDeviceAddr.....	83
CUDBGAPI_st::getPhysicalRegister30.....	84
CUDBGAPI_st::getPhysicalRegister40.....	85
CUDBGAPI_st::isDeviceCodeAddress.....	86
CUDBGAPI_st::isDeviceCodeAddress55.....	86
CUDBGAPI_st::lookupDeviceCodeSymbol.....	87
3.10. Events.....	87
CUDBGEvent.....	88
CUDBGEventCallbackData.....	88
CUDBGEventCallbackData40.....	88
CUDBGEventKind.....	88
CUDBGNotifyNewEventCallback.....	89
CUDBGNotifyNewEventCallback31.....	89
CUDBGAPI_st::acknowledgeEvent30.....	89
CUDBGAPI_st::acknowledgeEvents42.....	90
CUDBGAPI_st::acknowledgeSyncEvents.....	90
CUDBGAPI_st::getErrorStringEx.....	90
CUDBGAPI_st::getNextAsyncEvent50.....	91
CUDBGAPI_st::getNextAsyncEvent55.....	91
CUDBGAPI_st::getNextEvent.....	92
CUDBGAPI_st::getNextEvent30.....	92
CUDBGAPI_st::getNextEvent32.....	92
CUDBGAPI_st::getNextEvent42.....	93
CUDBGAPI_st::getNextSyncEvent50.....	93
CUDBGAPI_st::getNextSyncEvent55.....	94
CUDBGAPI_st::setNotifyNewEventCallback.....	94
CUDBGAPI_st::setNotifyNewEventCallback31.....	94
CUDBGAPI_st::setNotifyNewEventCallback40.....	95

3.3. Debugger API.....	95
<b>Chapter 4. Data Structures.....</b>	<b>96</b>
CUDBGEvent.....	96
cases.....	97
kind.....	97
CUDBGEvent::cases_st.....	97
contextCreate.....	98
contextDestroy.....	98
contextPop.....	98
contextPush.....	98
elfImageLoaded.....	98
internalError.....	98
kernelFinished.....	98
kernelReady.....	98
CUDBGEvent::cases_st::contextCreate_st.....	99
context.....	99
dev.....	99
tid.....	99
CUDBGEvent::cases_st::contextDestroy_st.....	99
context.....	99
dev.....	99
tid.....	99
CUDBGEvent::cases_st::contextPop_st.....	99
context.....	100
dev.....	100
tid.....	100
CUDBGEvent::cases_st::contextPush_st.....	100
context.....	100
dev.....	100
tid.....	100
CUDBGEvent::cases_st::elfImageLoaded_st.....	100
context.....	101
dev.....	101
handle.....	101
module.....	101
properties.....	101
size.....	101
CUDBGEvent::cases_st::internalError_st.....	101

errorType.....	101
CUDBGEvent::cases_st::kernelFinished_st.....	101
context.....	102
dev.....	102
function.....	102
functionEntry.....	102
gridId.....	102
module.....	102
tid.....	102
CUDBGEvent::cases_st::kernelReady_st.....	102
blockDim.....	103
context.....	103
dev.....	103
function.....	103
functionEntry.....	103
gridDim.....	103
gridId.....	103
module.....	103
parentGridId.....	103
tid.....	103
type.....	104
CUDBGEventCallbackData.....	104
tid.....	104
timeout.....	104
CUDBGEventCallbackData40.....	104
tid.....	104
CUDBGGridInfo.....	104
blockDim.....	105
context.....	105
dev.....	105
function.....	105
functionEntry.....	105
gridDim.....	105
gridId64.....	105
module.....	105
origin.....	105
parentGridId.....	105
tid.....	105

type.....	105
4.1. Debugger API.....	106
Chapter 5. Data Fields.....	107
Chapter 6. Deprecated List.....	117

---

# Chapter 1. Release Notes

## 1.1. 11.8 Release

### **New Unified Debugger backend**

A new debugger backend named the Unified Debugger (UD) has been introduced on Linux platforms with this release. UD is supported across multiple platforms including both Windows and Linux. The UD should mostly be transparent to existing clients of the API. The previous debugger backend, known as the classic debugger backend, can still be used by setting the environment variable `CUDBG_USE_LEGACY_DEBUGGER` to 1. UD is not supported on Maxwell GPUs. The clients of the API shall switch to the classic backend if Maxwell support is required.

### **Device side `cudaDeviceSynchronize()` undefined behavior**

The clients of the API shall prevent the use of `SingleStepWarp` in the deprecated `cudaDeviceSynchronize()` function. Instead, revert to stepping over the call with a BP set and resume.

### **`CUDBG_EVENT_KERNEL_READY` events are no longer delivered for GPU-launched grids**

`CUDBG_EVENT_KERNEL_READY` events for GPU-launched grids that were delivered over the `ASYNC` event pipe will no longer be sent. GPU-launched here refers to codes making use of CUDA Dynamic Parallelism. The existing implementation for this use case was imprecise. The callback did not report all GPU-launched grids before execution has begun, only those found on the device currently executing that were not previously reported during their launch. This functionality may be reintroduced in a future release. If this functionality is strictly required, the classic debugger backend can be used.

### **`getLoadedFunctionInfo`**

Added a new `getLoadedFunctionInfo` call to obtain the section number and address of loaded functions for a given module.

## 1.2. 12.3 Release

### **`disassemble()` deprecation notice**

The `disassemble()` API function is deprecated. It will be dropped in an upcoming release. API consumers should use the `nvdiasm` utility instead.

## 1.3. 7.0 Release

**Stability improvements. No API additions or changes.**

## 1.4. 6.5 Release

### **Predicate registers**

The per-thread predicate registers can be accessed and modified via the `readPredicates()` and `writePredicates()` calls. Each of these calls expects a buffer of sufficient size to cover all predicates for the current GPU architecture. The number of current predicate registers can be read back via the `getNumPredicates()` API call.

### **Condition code register**

The per-thread condition code register can be accessed and modified via the `readCCRegister()` and `writeCCRegister()` calls. The condition code register is a unsigned 32-bit register, whose format may vary by GPU architecture.

### **Device Name**

The `getDeviceName()` API returns a string containing the publically exposed product name of the GPU.

### **API Error Reporting Improvement**

The symbol `CUDBG_REPORT_DRIVER_API_ERROR_FLAGS` points to an unsigned 32-bit integer in the application's process space that controls API error reporting. The values that can be written into this flag are specified in the `CUDBGReportDriverApiErrorFlags` enum. In 6.5, setting the bit corresponding to `CUDBG_REPORT_DRIVER_API_ERROR_FLAGS_SUPPRESS_NOT_READY` in the variable `CUDBG_REPORT_DRIVER_API_ERROR_FLAGS` is supported. This will prevent CUDA API calls that return the runtime API error code `cudaErrorNotReady` or the driver API error code `cuErrorNotReady` from executing the CUDA API error reporting function.

---

# Chapter 2. Introduction

This document describes the API for the set routines and data structures available in the CUDA library to any debugger.

Starting with 3.0, the CUDA debugger API includes several major changes, of which only few are directly visible to end-users:

- ▶ Performance is greatly improved, both with respect to interactions with the debugger and the performance of applications being debugged.
- ▶ The format of cubins has changed to ELF and, as a consequence, most restrictions on debug compilations have been lifted. More information about the new object format is included below.

The debugger API has significantly changed, reflected in the CUDA-GDB sources.

## 2.1. Debugger API

The CUDA Debugger API was developed with the goal of adhering to the following principles:

- ▶ Policy free
- ▶ Explicit
- ▶ Axiomatic
- ▶ Extensible
- ▶ Machine oriented

Being explicit is another way of saying that we minimize the assumptions we make. As much as possible the API reflects machine state, not internal state.

There are two major "modes" of the devices: stopped or running. We switch between these modes explicitly with `suspendDevice` and `resumeDevice`, though the machine may suspend on its own accord, for example when hitting a breakpoint.

Only when stopped, can we query the machine's state. Warp state includes which function is it running, which block, which lanes are valid, etc.

As of CUDA 6.0, state collection functions in the debug API will return `CUDBG_ERROR_RUNNING_DEVICE` if called without first calling the `suspendDevice` entry point to ensure the device is stopped.

Clients of the debug API should suspend all devices before servicing a `CUDBGEvent`. A valid `CUDBGEvent` is only guaranteed to be returned after the notification callback set using `CUDBGAPI_st::setNotifyNewEventCallback()` is executed. Any debug API entry point will return `CUDBG_ERROR_RECURSIVE_API_CALL` when the call is made from within the notification callback set using `CUDBGAPI_st::setNotifyNewEventCallback()`.

## 2.2. ELF and DWARF

CUDA applications are compiled in ELF binary format.

Starting with CUDA 6.0, DWARF device information is obtained through an API call of `CUDBGAPI_st::getElfImageByHandle` using the handle exposed from `CUDBGEvent` of type `CUDBG_EVENT_ELF_IMAGE_LOADED`. This means that the information is not available until runtime, after the CUDA driver has loaded. The DWARF device information lifetime is valid until it is unloaded, which presents a `CUDBGEvent` of type `CUDBG_EVENT_ELF_IMAGE_UNLOADED`.

In CUDA 5.5 and earlier, the DWARF device information was returned as part of the `CUDBGEvent` of type `CUDBG_EVENT_ELF_IMAGE_LOADED`. The pointers presented in `CUDBGEvent55` were read-only pointers to memory managed by the debug API. The memory pointed to was implicitly scoped to the lifetime of the loading CUDA context. Accessing the returned pointers after the context was destroyed resulted in undefined behavior.

DWARF device information contains physical addresses for all device memory regions except for code memory. The address class field (`DW_AT_address_class`) is set for all device variables, and is used to indicate the memory segment type (`ptxStorageKind`). The physical addresses must be accessed using several segment-specific API calls.

For memory reads, see:

- ▶ `CUDBGAPI_st::readCodeMemory()`
- ▶ `CUDBGAPI_st::readConstMemory()`
- ▶ `CUDBGAPI_st::readGlobalMemory()`
- ▶ `CUDBGAPI_st::readParamMemory()`
- ▶ `CUDBGAPI_st::readSharedMemory()`
- ▶ `CUDBGAPI_st::readLocalMemory()`
- ▶ `CUDBGAPI_st::readTextureMemory()`

For memory writes, see:

- ▶ `CUDBGAPI_st::writeGlobalMemory()`
- ▶ `CUDBGAPI_st::writeParamMemory()`
- ▶ `CUDBGAPI_st::writeSharedMemory()`
- ▶ `CUDBGAPI_st::writeLocalMemory()`

Access to code memory requires a virtual address. This virtual address is embedded for all device code sections in the device ELF image. See the API call:

- ▶ `CUDBGAPI_st::readVirtualPC()`

Here is a typical DWARF entry for a device variable located in memory:

```
<2><321>: Abbrev Number: 18 (DW_TAG_formal_parameter)
  DW_AT_decl_file   : 27
  DW_AT_decl_line   : 5
  DW_AT_name        : res
  DW_AT_type         : <2c6>
  DW_AT_location    : 9 byte block: 3 18 0 0 0 0 0 0 0      (DW_OP_addr: 18)
  DW_AT_address_class: 7
```

The above shows that variable 'res' has an address class of 7 (`ptiParamStorage`). Its location information shows it is located at address 18 within the parameter memory segment.

Local variables are no longer spilled to local memory by default. The DWARF now contains variable-to-register mapping and liveness information for all variables. It can be the case that variables are spilled to local memory, and this is all contained in the DWARF information which is ULEB128 encoded (as a `DW_OP_regx` stack operation in the `DW_AT_location` attribute).

Here is a typical DWARF entry for a variable located in a local register:

```
<3><359>: Abbrev Number: 20 (DW_TAG_variable)
  DW_AT_decl_file   : 27
  DW_AT_decl_line   : 7
  DW_AT_name        : c
  DW_AT_type         : <1aa>
  DW_AT_location    : 7 byte block: 90 b9 e2 90 b3 d6 4      (DW_OP_regx:
160631632185)
  DW_AT_address_class: 2
```

This shows variable 'c' has address class 2 (`ptxRegStorage`) and its location can be found by decoding the ULEB128 value, `DW_OP_regx: 160631632185`. See `cuda-tdep.c` in the `cuda-gdb` source drop for information on decoding this value and how to obtain which physical register holds this variable during a specific device PC range.

Access to physical registers liveness information requires a 0-based physical PC. See the API call:

- ▶ `CUDBGAPI_st::readPC()`

## 2.3. ABI Support

ABI support is handled through the following thread API calls:

- ▶ `CUDBGAPI_st::readCallDepth()`
- ▶ `CUDBGAPI_st::readReturnAddress()`
- ▶ `CUDBGAPI_st::readVirtualReturnAddress()`

The return address is not accessible on the local stack and the API call must be used to access its value.

For more information, please refer to the ABI documentation titled "Fermi ABI: Application Binary Interface".

## 2.4. Exception Reporting

Some kernel exceptions are reported as device events and accessible via the API call:

- ▶ `CUDBGAPI_st::readLaneException()`

The reported exceptions are listed in the `CUDBGException_t` enum type. Each prefix, (Device, Warp, Lane), refers to the precision of the exception. That is, the lowest known execution unit that is responsible for the origin of the exception. All lane errors are precise; the exact instruction and lane that caused the error are known. Warp errors are typically within a few instructions of where the actual error occurred, but the exact lane within the warp is not known. On device errors, we *may* know the *kernel* that caused it. Explanations about each exception type can be found in the documentation of the struct.

Exception reporting is only supported on Fermi (sm\_20 or greater).

## 2.5. Attaching and Detaching

The debug client must take the following steps to attach to a running CUDA application:

1. Attach to the CPU process corresponding to the CUDA application. The CPU part of the application will be frozen at this point.
2. Check to see if the `CUDBG_IPC_FLAG_NAME` variable is accessible from the memory space of the application. If not, it implies that the application has not loaded the CUDA driver, and the attaching to the application is complete.
3. Make a dynamic (inferior) function call to the function `cudbgApiInit()` with an argument of "2", i.e., "`cudbgApiInit(2)`", e.g. by using `ptrace(2)` on Linux. This causes a helper process to be forked off from the application, which assists in attaching to the CUDA process.
4. Ensure that the initialization of the CUDA debug API is complete, or wait till API initialization is successful (i.e. call the "`initialize()`" API method until it succeeds).
5. Make the "`initializeAttachStub()`" API call to initialize the helper process that was forked off from the application earlier.
6. Read the value of the `CUDBG_RESUME_FOR_ATTACH_DETACH` variable from the memory space of the application:
  - ▶ If the value is non-zero, resume the CUDA application so that more data can be collected about the application and sent to the debugger. When the application is resumed, the debug client can expect to receive various CUDA events from the CUDA application. Once all state has been collected, the debug client will receive the event `CUDBG_EVENT_ATTACH_COMPLETE`.
  - ▶ If the value is zero, there is no more attach data to collect. Set the `CUDBG_IPC_FLAG_NAME` variable to 1 in the application's process space, which enables further events from the CUDA application.

7. At this point, attaching to the CUDA application is complete and all GPUs belonging to the CUDA application will be suspended.

The debug client must take the following steps to detach from a running CUDA application:

1. Check to see if the CUDBG\_IPC\_FLAG\_NAME variable is accessible from the memory space of the application, and that the CUDA debug API is initialized. If either of these conditions is not met, treat the application as CPU-only and detach from the application.
2. Next, make the "clearAttachState" API call to prepare the CUDA debug API for detach.
3. Make a dynamic (inferior) function call to the function cudbgApiDetach() in the memory space of the application, e.g. by using ptrace(2) on Linux. This causes CUDA driver to setup state for detach.
4. Read the value of the CUDBG\_RESUME\_FOR\_ATTACH\_DETACH variable from the memory space of the application. If the value is non-zero, make the "requestCleanupOnDetach" API call.
5. Set the CUDBG\_DEBUGGER\_INITIALIZED variable to 0 in the memory space of the application. This makes sure the debugger is reinitialized from scratch if the debug client re-attaches to the application in the future.
6. If the value of the CUDBG\_RESUME\_FOR\_ATTACH\_DETACH variable was found to be non-zero in step 4, delete all breakpoints and resume the CUDA application. This allows the CUDA driver to perform cleanups before the debug client detaches from it. Once the cleanup is complete, the debug client will receive the event CUDBG\_EVENT\_DETACH\_COMPLETE.
7. Set the CUDBG\_IPC\_FLAG\_NAME variable to zero in the memory space of the application. This prevents any more callbacks from the CUDA application to the debugger.
8. The client must then finalize the CUDA debug API.
9. Finally, detach from the CPU part of the CUDA application. At this point all GPUs belonging to the CUDA application will be resumed.

---

# Chapter 3. Modules

Here is a list of all modules:

- ▶ [General](#)
- ▶ [Initialization](#)
- ▶ [Device Execution Control](#)
- ▶ [Breakpoints](#)
- ▶ [Device State Inspection](#)
- ▶ [Device State Alteration](#)
- ▶ [Grid Properties](#)
- ▶ [Device Properties](#)
- ▶ [DWARF Utilities](#)
- ▶ [Events](#)

## 3.1. General

### enum CUDBGResult

Result values of all the API routines.

#### Values

**CUDBG\_SUCCESS = 0x0000**

The API call executed successfully.

**CUDBG\_ERROR\_UNKNOWN = 0x0001**

Error type not listed below.

**CUDBG\_ERROR\_BUFFER\_TOO\_SMALL = 0x0002**

Cannot copy all the queried data into the buffer argument.

**CUDBG\_ERROR\_UNKNOWN\_FUNCTION = 0x0003**

Function cannot be found in the CUDA kernel.

**CUDBG\_ERROR\_INVALID\_ARGS = 0x0004**

Wrong use of arguments (NULL pointer, illegal value,...).

**CUDBG\_ERROR\_UNINITIALIZED = 0x0005**

Debugger API has not yet been properly initialized.

**CUDBG\_ERROR\_INVALID\_COORDINATES = 0x0006**

Invalid block or thread coordinates were provided.

**CUDBG\_ERROR\_INVALID\_MEMORY\_SEGMENT = 0x0007**

Invalid memory segment requested.

**CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS = 0x0008**

Requested address (+size) is not within proper segment boundaries.

**CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED = 0x0009**

Memory is not mapped and cannot be mapped.

**CUDBG\_ERROR\_INTERNAL = 0x000a**

A debugger internal error occurred.

**CUDBG\_ERROR\_INVALID\_DEVICE = 0x000b**

Specified device cannot be found.

**CUDBG\_ERROR\_INVALID\_SM = 0x000c**

Specified sm cannot be found.

**CUDBG\_ERROR\_INVALID\_WARP = 0x000d**

Specified warp cannot be found.

**CUDBG\_ERROR\_INVALID\_LANE = 0x000e**

Specified lane cannot be found.

**CUDBG\_ERROR\_SUSPENDED\_DEVICE = 0x000f**

The requested operation is not allowed when the device is suspended.

**CUDBG\_ERROR\_RUNNING\_DEVICE = 0x0010**

Device is running and not suspended.

**CUDBG\_ERROR\_RESERVED\_0 = 0x0011****CUDBG\_ERROR\_INVALID\_ADDRESS = 0x0012**

Address is out-of-range.

**CUDBG\_ERROR\_INCOMPATIBLE\_API = 0x0013**

The requested API is not available.

**CUDBG\_ERROR\_INITIALIZATION\_FAILURE = 0x0014**

The API could not be initialized.

**CUDBG\_ERROR\_INVALID\_GRID = 0x0015**

The specified grid is not valid.

**CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE = 0x0016**

The event queue is empty and there is no event left to be processed.

**CUDBG\_ERROR\_SOME\_DEVICES\_WATCHDOGGED = 0x0017**

Some devices were excluded because they have a watchdog associated with them.

**CUDBG\_ERROR\_ALL\_DEVICES\_WATCHDOGGED = 0x0018**

All devices were excluded because they have a watchdog associated with them.

**CUDBG\_ERROR\_INVALID\_ATTRIBUTE = 0x0019**

Specified attribute does not exist or is incorrect.

**CUDBG\_ERROR\_ZERO\_CALL\_DEPTH = 0x001a**

No function calls have been made on the device.

**CUDBG\_ERROR\_INVALID\_CALL\_LEVEL = 0x001b**

Specified call level is invalid.

**CUDBG\_ERROR\_COMMUNICATION\_FAILURE = 0x001c**

Communication error between the debugger and the application.

**CUDBG\_ERROR\_INVALID\_CONTEXT = 0x001d**

Specified context cannot be found.

**CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM = 0x001e**

Requested address was not originally allocated from device memory (most likely visible in system memory).

**CUDBG\_ERROR\_MEMORY\_UNMAPPING\_FAILED = 0x001f**

Requested address is not mapped and cannot be unmapped.

**CUDBG\_ERROR\_INCOMPATIBLE\_DISPLAY\_DRIVER = 0x0020**

The display driver is incompatible with the API.

**CUDBG\_ERROR\_INVALID\_MODULE = 0x0021**

The specified module is not valid.

**CUDBG\_ERROR\_LANE\_NOT\_IN\_SYSCALL = 0x0022**

The specified lane is not inside a device syscall.

**CUDBG\_ERROR\_MEMCHECK\_NOT\_ENABLED = 0x0023**

Memcheck has not been enabled.

**CUDBG\_ERROR\_INVALID\_ENVVAR\_ARGS = 0x0024**

Some environment variable's value is invalid.

**CUDBG\_ERROR\_OS\_RESOURCES = 0x0025**

Error while allocating resources from the OS.

**CUDBG\_ERROR\_FORK\_FAILED = 0x0026**

Error while forking the debugger process.

**CUDBG\_ERROR\_NO\_DEVICE\_AVAILABLE = 0x0027**

No CUDA capable device was found.

**CUDBG\_ERROR\_ATTACH\_NOT\_POSSIBLE = 0x0028**

Attaching to the CUDA program is not possible.

**CUDBG\_ERROR\_WARP\_RESUME\_NOT\_POSSIBLE = 0x0029**

**CUDBG\_ERROR\_INVALID\_WARP\_MASK = 0x002a**

**CUDBG\_ERROR\_AMBIGUOUS\_MEMORY\_ADDRESS = 0x002b**

Specified device pointer cannot be resolved to a GPU unambiguously because it is valid on more than one GPU.

**CUDBG\_ERROR\_RECURSIVE\_API\_CALL = 0x002c**

**CUDBG\_ERROR\_MISSING\_DATA = 0x002d**

**CUDBG\_ERROR\_NOT\_SUPPORTED = 0x002e**

## 3.2. Initialization

## CUDBGResult (\*CUDBGAPI\_st::finalize) ()

Finalize the API and free all memory.

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_COMMUNICATION\_FAILURE, CUDBG\_ERROR\_UNKNOWN

Since CUDA 3.0.

See also:

[initialize](#)

## CUDBGResult (\*CUDBGAPI\_st::getSupportedDebuggerCapabilities) (CUDBGCapabilityFlags \*capabilities)

Returns debugger capabilities that are supported by this version of the API.

### Parameters

#### capabilities

- returned debugger capabilities

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,

Since CUDA 12.5.

This API method can be called without initializing the API.

## CUDBGResult (\*CUDBGAPI\_st::initialize) ()

Initialize the API.

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNKNOWN

Since CUDA 3.0.

See also:

[finalize](#)

## 3.4. Breakpoints

### CUDBGResult

**(\*CUDBGAPI\_st::getAdjustedCodeAddress) (uint32\_t devId, uint64\_t address, uint64\_t \*adjustedAddress, CUDBGAdjAddrAction adjAction)**

The client must call this function before inserting a breakpoint, or when the previous or next code address is needed. Returns the adjusted code address for a given code address for a given device.

#### Parameters

##### **devId**

- the device index

##### **address**

##### **adjustedAddress**

- adjusted address

##### **adjAction**

- whether the adjusted next, previous or current address is needed

#### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_INVALID\_ADDRESS, CUDBG\_ERROR\_INVALID\_DEVICE

Since CUDA 5.5.

See also:

[unsetBreakpoint](#)

**CUDBGResult (\*CUDBGAPI\_st::setBreakpoint) (uint32\_t dev, uint64\_t addr)**

Sets a breakpoint at the given instruction address for the given device. Before setting a breakpoint, CUDBGAPI\_st::getAdjustedCodeAddress should be called to get the adjusted breakpoint address.

#### Parameters

##### **dev**

- the device index

##### **addr**

- instruction address

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_INVALID\_ADDRESS, CUDBG\_ERROR\_INVALID\_DEVICE

Since CUDA 3.2.

See also:

[unsetBreakpoint](#)

## CUDBGResult (\*CUDBGAPI\_st::setBreakpoint31) (uint64\_t addr)

Sets a breakpoint at the given instruction address.

### Parameters

**addr**

- instruction address

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_INVALID\_ADDRESS

Since CUDA 3.0.

[Deprecated](#) in CUDA 3.2.

See also:

[unsetBreakpoint31](#)

## CUDBGResult (\*CUDBGAPI\_st::unsetBreakpoint) (uint32\_t dev, uint64\_t addr)

Unsets a breakpoint at the given instruction address for the given device.

### Parameters

**dev**

- the device index

**addr**

- instruction address

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_INVALID\_ADDRESS, CUDBG\_ERROR\_INVALID\_DEVICE

Since CUDA 3.2.

See also:

[setBreakpoint](#)

## CUDBGResult (\*CUDBGAPI\_st::unsetBreakpoint31) (uint64\_t addr)

Unsets a breakpoint at the given instruction address.

### Parameters

**addr**

- instruction address

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

[Deprecated](#) in CUDA 3.2.

See also:

[setBreakpoint31](#)

## 3.5. Device State Inspection

### CUDBGResult (\*CUDBGAPI\_st::generateCoredump) (const char \*filename, CUDBGCoreDumpGenerationFlags flags)

Generates a coredump for the current GPU state.

### Parameters

**filename**

- target coredump file name

**flags**

- coredump generation flags/options

**Returns**

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_NOT\_SUPPORTED

Since CUDA 12.3

**CUDBGResult (\*CUDBGAPI\_st::getConstBankAddress)**  
**(uint32\_t dev, uint64\_t gridId64, uint32\_t bank, uint64\_t**  
**\*address, uint32\_t \*size)**

Get constant bank GPU VA and size.

**Parameters****dev**

- device index

**gridId64**

- grid ID of the grid containing the constant bank

**bank**

- constant bank number

**address**

- (output) GPU VA of the bank memory

**size**

- (output) bank size

**Returns**

CUDBG\_ERROR\_NOT\_SUPPORTED

Since CUDA 12.4

**CUDBGResult**  
**(\*CUDBGAPI\_st::getConstBankAddress123) (uint32\_t**  
**dev, uint32\_t sm, uint32\_t wp, uint32\_t bank, uint32\_t**  
**offset, uint64\_t \*address)**

Convert constant bank number and offset into GPU VA.

**Parameters****dev**

- device index

**sm**

- SM index

**wp**

- warp index

**bank**

- constant bank number

**offset**

- offset within the bank

**address**

- (output) GPU VA

**Returns**

CUDBG\_ERROR\_NOT\_SUPPORTED

Since CUDA 12.3

**CUDBGResult (\*CUDBGAPI\_st::getLoadedFunctionInfo)**  
**(uint32\_t devId, uint64\_t handle,**  
**CUDBGLoadedFunctionInfo \*info, uint32\_t startIndex,**  
**uint32\_t numEntries)**

Get the section number and address of loaded functions for a given module.

**Parameters****devId****handle**

- ELF/cubin image handle

**info****startIndex****numEntries**

- number of function load entries to read

**Returns**

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_NOT\_SUPPORTED

Since CUDA 11.8

## CUDBGResult

(\*CUDBGAPI\_st::getLoadedFunctionInfo118) (uint32\_t devId, uint64\_t handle, CUDBGLoadedFunctionInfo \*info, uint32\_t numEntries)

Get the section number and address of loaded functions for a given module.

### Parameters

**devId**

**handle**

- ELF/cubin image handle

**info**

**numEntries**

- number of function load entries to read

### Returns

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_NOT\_SUPPORTED

Since CUDA 11.8

## CUDBGResult

(\*CUDBGAPI\_st::getManagedMemoryRegionInfo) (uint64\_t startAddress, CUDBGMemoryInfo \*memoryInfo, uint32\_t memoryInfo\_size, uint32\_t \*numEntries)

Returns a sorted list of managed memory regions The sorted list of memory regions starts from a region containing the specified starting address. If the starting address is set to 0, a sorted list of managed memory regions is returned which starts from the managed memory region with the lowest start address.

### Parameters

**startAddress**

- The address that the first region in the list must contain. If the starting address is set to 0, the list of managed memory regions returned starts from the managed memory region with the lowest start address.

**memoryInfo**

- Client-allocated array of memory region records of type CUDBGMemoryInfo.

**memoryInfo\_size**

- Number of records of type CUDBGMemoryInfo that memoryInfo can hold.

**numEntries**

- Pointer to a client-allocated variable holding the number of valid entries returned in memoryInfo. Valid entries are contiguous and start from memoryInfo[0].

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,  
CUDBG\_ERROR\_INVALID\_ADDRESS, CUDBG\_ERROR\_INTERNAL

Since CUDA 6.0.

**CUDBGResult**

(\*CUDBGAPI\_st::memcheckReadErrorAddress) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint64\_t \*address, ptxStorageKind \*storage)

Get the address that memcheck detected an error on.

**Parameters****dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**address**

- returned address detected by memcheck

**storage**

- returned address class of address

**Returns**

CUDBG\_ERROR\_NOT\_SUPPORTED,

Since CUDA 5.0.

## CUDBGResult (\*CUDBGAPI\_st::readActiveLanes) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*activeLanesMask)

Reads the bitmask of active lanes on a valid warp.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**activeLanesMask**

- the returned bitmask of active lanes

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[readGridId](#)

[readBlockIdx](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

## CUDBGResult

(\*CUDBGAPI\_st::readAllVirtualReturnAddresses)  
 (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,  
 uint64\_t \*addrs, uint32\_t numAddrs, uint32\_t \*callDepth,  
 uint32\_t \*syscallCallDepth)

Reads all the virtual return addresses.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**addrs**

- the returned addresses array

**numAddrs**

- number of elements in addrs array

**callDepth**

- the returned call depth

**syscallCallDepth**

- the returned syscall call depth

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_GRID,  
 CUDBG\_ERROR\_UNKNOWN\_FUNCTION, CUDBG\_ERROR\_UNINITIALIZED,  
 CUDBG\_ERROR\_INTERNAL

Since CUDA 12.5.

See also:

[readCallDepth](#)

[readReturnAddress](#)

## CUDBGResult (\*CUDBGAPI\_st::readBlockIdx) (uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim3 \*blockIdx)

Reads the CUDA block index running on a valid warp.

### Parameters

#### **dev**

- device index

#### **sm**

- SM index

#### **wp**

- warp index

#### **blockIdx**

- the returned CUDA block index

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 4.0.

See also:

[readGridId](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*CUDBGAPI\_st::readBlockIdx32) (uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim2 \*blockIdx)

Reads the two-dimensional CUDA block index running on a valid warp.

### Parameters

#### **dev**

- device index

**sm**

- SM index

**wp**

- warp index

**blockIdx**

- the returned CUDA block index

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

Deprecated in CUDA 4.0.

See also:

[readGridId](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*CUDBGAPI\_st::readBrokenWarps) (uint32\_t dev, uint32\_t sm, uint64\_t \*brokenWarpsMask)

Reads the bitmask of warps that are at a breakpoint on a given SM.

**Parameters****dev**

- device index

**sm**

- SM index

**brokenWarpsMask**

- the returned bitmask of broken warps

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[readGridId](#)

[readBlockIdx](#)

[readThreadIdx](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

**CUDBGResult (\*CUDBGAPI\_st::readCallDepth) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint32\_t \*depth)**

Reads the call depth (number of calls) for a given lane.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**depth**

- the returned call depth

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 4.0.

See also:

[readReturnAddress](#)

[readVirtualReturnAddress](#)

## CUDBGResult (\*CUDBGAPI\_st::readCallDepth32) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*depth)

Reads the call depth (number of calls) for a given warp.

### Parameters

#### **dev**

- device index

#### **sm**

- SM index

#### **wp**

- warp index

#### **depth**

- the returned call depth

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.1.

Deprecated in CUDA 4.0.

See also:

[readReturnAddress32](#)

[readVirtualReturnAddress32](#)

## CUDBGResult (\*CUDBGAPI\_st::readCCRegister) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint32\_t \*val)

Reads the hardware CC register.

### Parameters

#### **dev**

- device index

#### **sm**

- SM index

#### **wp**

- warp index

**ln**

- lane index

**val**

- buffer

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 6.5.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readGlobalMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

[readPredicates](#)

**CUDBGResult (\*CUDBGAPI\_st::readClusterIdx)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim3**  
**\*clusterIdx)**

Reads the CUDA cluster index running on a valid warp.

**Parameters****dev**

- device index

**sm**

- SM index

**wp**

- warp index

**clusterIdx**

- the returned CUDA cluster index

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 12.0.

See also:

[readGridId](#)

[readThreadIdx](#)

[readBlockIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*CUDBGAPI\_st::readCodeMemory) (uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)

Reads content at address in the code memory segment.

**Parameters****dev**

- device index

**addr**

- memory address

**buf**

- buffer

**sz**

- size of the buffer

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

See also:

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

## CUDBGResult (\*CUDBGAPI\_st::readConstMemory) (uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)

Reads content at address in the constant memory segment.

### Parameters

#### **dev**

- device index

#### **addr**

- memory address

#### **buf**

- buffer

#### **sz**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

See also:

[readCodeMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*CUDBGAPI\_st::readErrorPC) (uint32\_t devId, uint32\_t sm, uint32\_t wp, uint64\_t \*errorPC, bool \*errorPCValid)**

Get the hardware reported error PC if it exists.

### Parameters

**devId**

- the device index

**sm**

- the SM index

**wp****errorPC**

- PC of the exception

**errorPCValid**

- boolean to indicate that the returned error PC is valid

### Returns

CUDBG\_SUCCESS CUDBG\_ERROR\_UNINITIALIZED CUDBG\_ERROR\_INVALID\_DEVICE  
CUDBG\_ERROR\_INVALID\_SM CUDBG\_ERROR\_INVALID\_WARP  
CUDBG\_ERROR\_INVALID\_ARGS CUDBG\_ERROR\_UNKNOWN\_FUNCTION

Since CUDA 6.0

**CUDBGResult (\*CUDBGAPI\_st::readGenericMemory) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint64\_t addr, void \*buf, uint32\_t sz)**

Reads content at an address in the generic address space. This function determines if the given address falls into the local, shared, or global memory window. It then accesses memory taking into account the hardware co-ordinates provided as inputs.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**addr**

- memory address

**buf**

- buffer

**sz****Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,  
 CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE,  
 CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED,  
 CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM

Since CUDA 6.0.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

## CUDBGResult (\*CUDBGAPI\_st::readGlobalMemory) (uint64\_t addr, void \*buf, uint32\_t sz)

Reads content at an address in the global address space. If the address is valid on more than one device and one of those devices does not support UVA, an error is returned.

**Parameters****addr**

- memory address

**buf**

- buffer

**SZ**

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED,  
 CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS,  
 CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM  
 CUDBG\_ERROR\_AMBIGUOUS\_MEMORY\_ADDRESS\_

Since CUDA 6.0.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

## CUDBGResult (\*CUDBGAPI\_st::readGlobalMemory31) (uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)

Reads content at address in the global memory segment.

### Parameters

**dev**

- device index

**addr**

- memory address

**buf**

- buffer

**SZ**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

Deprecated in CUDA 3.2.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*CUDBGAPI\_st::readGlobalMemory55)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,**  
**uint64\_t addr, void \*buf, uint32\_t sz)**

Reads content at address in the global memory segment (entire 40-bit VA on Fermi+).

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**addr**

- memory address

**buf**

- buffer

**sz**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,  
 CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE,

CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED,  
 CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM

Since CUDA 3.2.

Deprecated in CUDA 6.0.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*CUDBGAPI\_st::readGridId) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t \*gridId64)**

Reads the 64-bit CUDA grid index running on a valid warp.

### Parameters

#### **dev**

- device index

#### **sm**

- SM index

#### **wp**

- warp index

#### **gridId64**

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 5.5.

See also:

[readBlockIdx](#)[readThreadIdx](#)[readBrokenWarps](#)[readValidWarps](#)[readValidLanes](#)[readActiveLanes](#)

**CUDBGResult (\*CUDBGAPI\_st::readGridId50) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*gridId)**

Reads the CUDA grid index running on a valid warp.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**gridId**

- the returned CUDA grid index

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

Deprecated in CUDA 5.5.

See also:

[readBlockIdx](#)[readThreadIdx](#)[readBrokenWarps](#)[readValidWarps](#)[readValidLanes](#)[readActiveLanes](#)

**CUDBGResult (\*CUDBGAPI\_st::readLaneException)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,**  
**CUDBGException\_t \*exception)**

Reads the exception type for a given lane.

#### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**exception**

- the returned exception type

#### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM,  
 CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.1.

**CUDBGResult (\*CUDBGAPI\_st::readLaneStatus)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, bool**  
**\*error)**

Reads the status of the given lane. For specific error values, use readLaneException.

#### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**error**

- true if there is an error

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

**CUDBGResult (\*CUDBGAPI\_st::readLocalMemory)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,**  
**uint64\_t addr, void \*buf, uint32\_t sz)**

Reads content at address in the local memory segment.

**Parameters****dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**addr**

- memory address

**buf**

- buffer

**sz**

- size of the buffer

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

See also:

[readCodeMemory](#)

[readConstMemory](#)[readGenericMemory](#)[readParamMemory](#)[readSharedMemory](#)[readTextureMemory](#)[readRegister](#)[readPC](#)

**CUDBGResult** (\*CUDBGAPI\_st::readParamMemory)  
 (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t addr, void  
 \*buf, uint32\_t sz)

Reads content at address in the param memory segment.

### Parameters

#### **dev**

- device index

#### **sm**

- SM index

#### **wp**

- warp index

#### **addr**

- memory address

#### **buf**

- buffer

#### **sz**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

See also:

[readCodeMemory](#)[readConstMemory](#)[readGenericMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*CUDBGAPI\_st::readPC) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint64\_t \*pc)**

Reads the PC on the given active lane.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**pc**

- the returned PC

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNKNOWN\_FUNCTION, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readVirtualPC](#)

## CUDBGResult (\*CUDBGAPI\_st::readPinnedMemory) (uint64\_t addr, void \*buf, uint32\_t sz)

Reads content at pinned address in system memory.

### Parameters

#### **addr**

- system memory address

#### **buf**

- buffer

#### **sz**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,  
CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.2.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult** (\*CUDBGAPI\_st::readPredicates) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint32\_t predicates\_size, uint32\_t \*predicates)

Reads content of hardware predicate registers.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**predicates\_size**

- number of predicate registers to read

**predicates**

- buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 6.5.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readGlobalMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult** (\*CUDBGAPI\_st::readRegister) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint32\_t regno, uint32\_t \*val)

Reads content of a hardware register.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**regno**

- register index

**val**

- buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readPC](#)

**CUDBGResult** (\*CUDBGAPI\_st::readRegisterRange)  
 (uint32\_t devId, uint32\_t sm, uint32\_t wp, uint32\_t ln,  
 uint32\_t index, uint32\_t registers\_size, uint32\_t \*registers)

Reads content of a hardware range of hardware registers.

### Parameters

**devId**

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**index**

- index of the first register to read

**registers\_size**

- number of registers to read

**registers**

- buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM,  
 CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 6.0.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readPC](#)

[readRegister](#)

**CUDBGResult (\*CUDBGAPI\_st::readReturnAddress)**  
(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,  
uint32\_t level, uint64\_t \*ra)

Reads the physical return address for a call level.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**level**

- the specified call level

**ra**

- the returned return address for level

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_GRID,  
CUDBG\_ERROR\_INVALID\_CALL\_LEVEL, CUDBG\_ERROR\_ZERO\_CALL\_DEPTH,  
CUDBG\_ERROR\_UNKNOWN\_FUNCTION, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 4.0.

See also:

[readCallDepth](#)

[readVirtualReturnAddress](#)

**CUDBGResult** (\*CUDBGAPI\_st::readReturnAddress32)  
(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t level,  
uint64\_t \*ra)

Reads the physical return address for a call level.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**level**

- the specified call level

**ra**

- the returned return address for level

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_INVALID\_CALL\_LEVEL,  
CUDBG\_ERROR\_ZERO\_CALL\_DEPTH, CUDBG\_ERROR\_UNKNOWN\_FUNCTION,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.1.

Deprecated in CUDA 4.0.

See also:

[readCallDepth32](#)

[readVirtualReturnAddress32](#)

**CUDBGResult** (\*CUDBGAPI\_st::readSharedMemory)  
(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t addr, void \*buf, uint32\_t sz)

Reads content at address in the shared memory segment.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**addr**

- memory address

**buf**

- buffer

**sz**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readLocalMemory](#)

[readTextureMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*CUDBGAPI\_st::readSmException)**  
**(uint32\_t dev, uint32\_t sm, CUDBGException\_t**  
**\*exception, uint64\_t \*errorPC, bool \*errorPCValid)**

Get the SM exception status if it exists.

### Parameters

#### **dev**

- the device index

#### **sm**

- the SM index

#### **exception**

- returned exception

#### **errorPC**

- returned PC of the exception

#### **errorPCValid**

- boolean to indicate that the returned error PC is valid

### Returns

CUDBG\_SUCCESS CUDBG\_ERROR\_UNINITIALIZED CUDBG\_ERROR\_INVALID\_DEVICE  
 CUDBG\_ERROR\_INVALID\_SM CUDBG\_ERROR\_INVALID\_ARGS  
 CUDBG\_ERROR\_UNKNOWN\_FUNCTION

Since CUDA 12.5.

**CUDBGResult (\*CUDBGAPI\_st::readSyscallCallDepth)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,**  
**uint32\_t \*depth)**

Reads the call depth of syscalls for a given lane.

### Parameters

#### **dev**

- device index

#### **sm**

- SM index

#### **wp**

- warp index

#### **ln**

- lane index

**depth**

- the returned call depth

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 4.1.

See also:

[readReturnAddress](#)

[readVirtualReturnAddress](#)

**CUDBGResult (\*CUDBGAPI\_st::readTextureMemory)**  
**(uint32\_t devId, uint32\_t vsm, uint32\_t wp, uint32\_t id,**  
**uint32\_t dim, uint32\_t \*coords, void \*buf, uint32\_t sz)**

This method is no longer supported since CUDA 12.0.

**Parameters****devId**

- device index

**vsm**

- SM index

**wp**

- warp index

**id**

- texture id (the value of DW\_AT\_location attribute in the relocated ELF image)

**dim**

- texture dimension (1 to 4)

**coords**

- array of coordinates of size dim

**buf**

- result buffer

**sz**

- size of the buffer

**Returns**

CUDBG\_ERROR\_NOT\_SUPPORTED,

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

## CUDBGResult

(\*CUDBGAPI\_st::readTextureMemoryBindless)

(uint32\_t devId, uint32\_t vsm, uint32\_t wp, uint32\_t texSymtabIndex, uint32\_t dim, uint32\_t \*coords, void \*buf, uint32\_t sz)

This method is no longer supported since CUDA 12.0.

### Parameters

#### **devId**

- device index

#### **vsm**

- SM index

#### **wp**

- warp index

#### **texSymtabIndex**

- global symbol table index of the texture symbol

#### **dim**

- texture dimension (1 to 4)

#### **coords**

- array of coordinates of size dim

#### **buf**

- result buffer

#### **sz**

- size of the buffer

### Returns

CUDBG\_ERROR\_NOT\_SUPPORTED

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*CUDBGAPI\_st::readThreadIdx)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,**  
**CuDim3 \*threadIdx)**

Reads the CUDA thread index running on valid lane.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**threadIdx**

- the returned CUDA thread index

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM,  
 CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[readGridId](#)

[readBlockIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

**CUDBGResult (\*CUDBGAPI\_st::readUniformPredicates)**  
(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t  
predicates\_size, uint32\_t \*predicates)

Reads contents of uniform predicate registers.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**predicates\_size**

- number of predicate registers to read

**predicates**

- buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 10.0.

See also:

[readPredicates](#)

## CUDBGResult

(\*CUDBGAPI\_st::readUniformRegisterRange) (uint32\_t devId, uint32\_t sm, uint32\_t wp, uint32\_t regno, uint32\_t registers\_size, uint32\_t \*registers)

Reads a range of uniform registers.

### Parameters

**devId**

**sm**

- SM index

**wp**

- warp index

**regno**

- starting index into uniform register file

**registers\_size**

- number of bytes to read

**registers**

- pointer to buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 10.0.

See also:

[readRegister](#)

CUDBGResult (\*CUDBGAPI\_st::readValidLanes) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*validLanesMask)

Reads the bitmask of valid lanes on a given warp.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**validLanesMask**

- the returned bitmask of valid lanes

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[readGridId](#)

[readBlockIdx](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readActiveLanes](#)

## CUDBGResult (\*CUDBGAPI\_st::readValidWarps) (uint32\_t dev, uint32\_t sm, uint64\_t \*validWarpsMask)

Reads the bitmask of valid warps on a given SM.

**Parameters****dev**

- device index

**sm**

- SM index

**validWarpsMask**

- the returned bitmask of valid warps

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[readGridId](#)

[readBlockIdx](#)

[readThreadId](#)

[readBrokenWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

**CUDBGResult** (\*CUDBGAPI\_st::readVirtualPC) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint64\_t \*pc)

Reads the virtual PC on the given active lane.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**pc**

- the returned PC

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_UNKNOWN\_FUNCTION

Since CUDA 3.0.

See also:

[readPC](#)

## CUDBGResult

(\*CUDBGAPI\_st::readVirtualReturnAddress) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint32\_t level, uint64\_t \*ra)

Reads the virtual return address for a call level.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**level**

- the specified call level

**ra**

- the returned virtual return address for level

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_INVALID\_CALL\_LEVEL, CUDBG\_ERROR\_ZERO\_CALL\_DEPTH, CUDBG\_ERROR\_UNKNOWN\_FUNCTION, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_INTERNAL

Since CUDA 4.0.

See also:

[readCallDepth](#)

[readReturnAddress](#)

## CUDBGResult

(\*CUDBGAPI\_st::readVirtualReturnAddress32) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t level, uint64\_t \*ra)

Reads the virtual return address for a call level.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**level**

- the specified call level

**ra**

- the returned virtual return address for level

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_INVALID\_CALL\_LEVEL, CUDBG\_ERROR\_ZERO\_CALL\_DEPTH, CUDBG\_ERROR\_UNKNOWN\_FUNCTION, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_INTERNAL

Since CUDA 3.1.

Deprecated in CUDA 4.0.

See also:

[readCallDepth32](#)

[readReturnAddress32](#)

**CUDBGResult (\*CUDBGAPI\_st::readWarpState)**  
(uint32\_t dev, uint32\_t sm, uint32\_t wp,  
CUDBGWarpState \*state)

Get state of a given warp.

#### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**state**

- pointer to structure that contains warp status

#### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_UNINITIALIZED,

Since CUDA 12.0.

**CUDBGResult (\*CUDBGAPI\_st::readWarpState60)**  
(uint32\_t devId, uint32\_t sm, uint32\_t wp,  
CUDBGWarpState60 \*state)

Get state of a given warp.

#### Parameters

**devId****sm**

- SM index

**wp**

- warp index

**state**

- pointer to structure that contains warp status

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_UNINITIALIZED,

Since CUDA 6.0.

## CUDBGResult (\*CUDBGAPI\_st::writePinnedMemory) (uint64\_t addr, const void \*buf, uint32\_t sz)

Writes content to pinned address in system memory.

### Parameters

**addr**

- system memory address

**buf**

- buffer

**sz**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,  
 CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.2.

See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult** (\*CUDBGAPI\_st::writePredicates)  
(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,  
uint32\_t predicates\_size, const uint32\_t \*predicates)

Writes content to hardware predicate registers.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**predicates\_size**

- number of predicate registers to write

**predicates**

- buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM,  
CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 6.5.

See also:

[writeConstMemory](#)

[writeGenericMemory](#)

[writeGlobalMemory](#)

[writeParamMemory](#)

[writeSharedMemory](#)

[writeTextureMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

**CUDBGResult** (\*CUDBGAPI\_st::writeUniformPredicates)  
(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t  
predicates\_size, const uint32\_t \*predicates)

Writes to uniform predicate registers.

#### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**predicates\_size**

- number of predicate registers to write

**predicates**

- buffer

#### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 10.0.

See also:

`readUniformPredicate`

[writeRegister](#)

## 3.6. Device State Alteration

## CUDBGResult (\*CUDBGAPI\_st::writeCCRegister) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint32\_t val)

Writes the hardware CC register.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**val**

- value to write to the CC register

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM,  
CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 6.5.

See also:

[writeConstMemory](#)

[writeGenericMemory](#)

[writeGlobalMemory](#)

[writeParamMemory](#)

[writeSharedMemory](#)

[writeTextureMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

[writePredicates](#)

**CUDBGResult (\*CUDBGAPI\_st::writeGenericMemory)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,**  
**uint64\_t addr, const void \*buf, uint32\_t sz)**

Writes content to an address in the generic address space. This function determines if the given address falls into the local, shared, or global memory window. It then accesses memory taking into account the hardware co-ordinates provided as inputs.

### Parameters

#### **dev**

- device index

#### **sm**

- SM index

#### **wp**

- warp index

#### **ln**

- lane index

#### **addr**

- memory address

#### **buf**

- buffer

#### **sz**

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,  
 CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE,  
 CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED,  
 CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM

Since CUDA 6.0.

See also:

[writeParamMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

## CUDBGResult (\*CUDBGAPI\_st::writeGlobalMemory) (uint64\_t addr, const void \*buf, uint32\_t sz)

Writes content to an address in the global address space. If the address is valid on more than one device and one of those devices does not support UVA, an error is returned.

### Parameters

#### **addr**

- memory address

#### **buf**

- buffer

#### **sz**

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED,  
CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS,  
CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM  
CUDBG\_ERROR\_AMBIGUOUS\_MEMORY\_ADDRESS\_

Since CUDA 6.0.

See also:

[writeParamMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

## CUDBGResult (\*CUDBGAPI\_st::writeGlobalMemory31) (uint32\_t dev, uint64\_t addr, const void \*buf, uint32\_t sz)

Writes content to address in the global memory segment.

### Parameters

#### **dev**

- device index

#### **addr**

- memory address

#### **buf**

- buffer

**sz**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM,  
 CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED,  
 CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

Deprecated in CUDA 3.2.

See also:

[writeParamMemory](#)[writeSharedMemory](#)[writeLocalMemory](#)[writeRegister](#)

**CUDBGResult (\*CUDBGAPI\_st::writeGlobalMemory55)**  
 (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,  
 uint64\_t addr, const void \*buf, uint32\_t sz)

Writes content to address in the global memory segment (entire 40-bit VA on Fermi+).

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**addr**

- memory address

**buf**

- buffer

**sz**

- size of the buffer

## Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,  
 CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE,  
 CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED,  
 CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM

Since CUDA 3.2.

Deprecated in CUDA 6.0.

See also:

[writeParamMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

**CUDBGResult (\*CUDBGAPI\_st::writeLocalMemory)**  
 (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln,  
 uint64\_t addr, const void \*buf, uint32\_t sz)

Writes content to address in the local memory segment.

## Parameters

### **dev**

- device index

### **sm**

- SM index

### **wp**

- warp index

### **ln**

- lane index

### **addr**

- memory address

### **buf**

- buffer

### **sz**

- size of the buffer

## Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

See also:

[writeGenericMemory](#)

[writeParamMemory](#)

[writeSharedMemory](#)

[writeRegister](#)

**CUDBGResult (\*CUDBGAPI\_st::writeParamMemory)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t addr,**  
**const void \*buf, uint32\_t sz)**

Writes content to address in the param memory segment.

## Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**addr**

- memory address

**buf**

- buffer

**sz**

- size of the buffer

## Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

See also:

[writeGenericMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

**CUDBGResult (\*CUDBGAPI\_st::writeRegister) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint32\_t regno, uint32\_t val)**

Writes content to a hardware register.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**ln**

- lane index

**regno**

- register index

**val**

- buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[writeGenericMemory](#)

[writeParamMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

**CUDBGResult** (\*CUDBGAPI\_st::writeSharedMemory)  
(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t addr,  
const void \*buf, uint32\_t sz)

Writes content to address in the shared memory segment.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**addr**

- memory address

**buf**

- buffer

**sz**

- size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED

Since CUDA 3.0.

See also:

[writeGenericMemory](#)

[writeParamMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

**CUDBGResult** (\*CUDBGAPI\_st::writeUniformRegister)  
(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t regno,  
uint32\_t val)

Writes content to a uniform register.

#### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**regno**

- register index

**val**

- buffer

#### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 10.0.

See also:

[writeRegister](#)

[readUniformRegisterRange](#)

## 3.7. Grid Properties

**struct CUDBGGridInfo**

Grid info.

**enum CUDBGGridStatus**

Grid status.

## Values

### **CUDBG\_GRID\_STATUS\_INVALID**

An invalid grid ID was passed, or an error occurred during status lookup.

### **CUDBG\_GRID\_STATUS\_PENDING**

The grid was launched but is not running on the HW yet.

### **CUDBG\_GRID\_STATUS\_ACTIVE**

The grid is currently running on the HW.

### **CUDBG\_GRID\_STATUS\_SLEEPING**

The grid is on the device, doing a join.

### **CUDBG\_GRID\_STATUS\_TERMINATED**

The grid has finished executing.

### **CUDBG\_GRID\_STATUS\_UNDETERMINED**

The grid is either QUEUED or TERMINATED.

## CUDBGResult (\*CUDBGAPI\_st::getBlockDim) (uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim3 \*blockDim)

Get the number of threads in the given block.

## Parameters

### **dev**

- device index

### **sm**

- SM index

### **wp**

- warp index

### **blockDim**

- the returned number of threads in the block

## Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[getGridDim](#)

## CUDBGResult (\*CUDBGAPI\_st::getClusterDim) (uint32\_t dev, uint64\_t gridId64, CuDim3 \*clusterDim)

Get the number of blocks in the given cluster.

### Parameters

#### **dev**

- device index

#### **gridId64**

- grid ID

#### **clusterDim**

- the returned number of blocks in the cluster

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 12.0.

See also:

[getBlockDim](#)

[getGridDim](#)

## CUDBGResult (\*CUDBGAPI\_st::getElfImage) (uint32\_t dev, uint32\_t sm, uint32\_t wp, bool relocated, void \*\*elfImage, uint64\_t \*size)

Get the relocated or non-relocated ELF image and size for the grid on the given device.

### Parameters

#### **dev**

- device index

#### **sm**

- SM index

#### **wp**

- warp index

#### **relocated**

- set to true to specify the relocated ELF image, false otherwise

#### **\*elfImage**

- pointer to the ELF image

**size**

- size of the ELF image (64 bits)

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 4.0.

**CUDBGResult (\*CUDBGAPI\_st::getElfImage32)**  
**(uint32\_t dev, uint32\_t sm, uint32\_t wp, bool relocated,**  
**void \*\*elfImage, uint32\_t \*size)**

Get the relocated or non-relocated ELF image and size for the grid on the given device.

**Parameters****dev**

- device index

**sm**

- SM index

**wp**

- warp index

**relocated**

- set to true to specify the relocated ELF image, false otherwise

**\*elfImage**

- pointer to the ELF image

**size**

- size of the ELF image (32 bits)

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

Deprecated in CUDA 4.0.

**CUDBGResult (\*CUDBGAPI\_st::getGridAttribute)**  
 (uint32\_t dev, uint32\_t sm, uint32\_t wp, CUDBGAttribute  
 attr, uint64\_t \*value)

Get the value of a grid attribute.

#### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**attr**

- the attribute

**value**

- the returned value of the attribute

#### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID,  
 CUDBG\_ERROR\_INVALID\_ATTRIBUTE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.1.

**CUDBGResult (\*CUDBGAPI\_st::getGridAttributes)**  
 (uint32\_t dev, uint32\_t sm, uint32\_t wp,  
 CUDBGAttributeValuePair \*pairs, uint32\_t numPairs)

Get several grid attribute values in a single API call.

#### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**pairs**

- array of attribute/value pairs

**numPairs**

- the number of attribute/values pairs in the array

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_INVALID\_ATTRIBUTE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.1.

**CUDBGResult (\*CUDBGAPI\_st::getGridDim) (uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim3 \*gridDim)**

Get the number of blocks in the given grid.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**gridDim**

- the returned number of blocks in the grid

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 4.0.

See also:

[getBlockDim](#)

**CUDBGResult (\*CUDBGAPI\_st::getGridDim32) (uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim2 \*gridDim)**

Get the number of blocks in the given grid.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**gridDim**

- the returned number of blocks in the grid

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

Deprecated in CUDA 4.0.

See also:

[getBlockDim](#)

## CUDBGResult (\*CUDBGAPI\_st::getGridInfo) (uint32\_t dev, uint64\_t gridId64, CUDBGGridInfo \*gridInfo)

Get information about the specified grid. If the context of the grid has already been destroyed, the function will return CUDBG\_ERROR\_INVALID\_GRID, although the grid id is correct.

**Parameters****dev****gridId64****gridInfo**

- pointer to a client allocated structure in which grid info will be returned.

**Returns**

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_SUCCESS

Since CUDA 12.0.

## CUDBGResult (\*CUDBGAPI\_st::getGridInfo55) (uint32\_t dev, uint64\_t gridId64, CUDBGGridInfo55 \*gridInfo)

Get information about the specified grid. If the context of the grid has already been destroyed, the function will return CUDBG\_ERROR\_INVALID\_GRID, although the grid id is correct.

### Parameters

**dev**

**gridId64**

**gridInfo**

- pointer to a client allocated structure in which grid info will be returned.

### Returns

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_SUCCESS

Since CUDA 5.5.

## CUDBGResult (\*CUDBGAPI\_st::getGridStatus) (uint32\_t dev, uint64\_t gridId64, CUDBGGridStatus \*status)

Check whether the grid corresponding to the given gridId is still present on the device.

### Parameters

**dev**

**gridId64**

- 64-bit grid ID

**status**

- enum indicating whether the grid status is INVALID, PENDING, ACTIVE, SLEEPING, TERMINATED or UNDETERMINED

### Returns

CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_GRID, CUDBG\_ERROR\_INTERNAL

Since CUDA 5.5.

## CUDBGResult (\*CUDBGAPI\_st::getGridStatus50) (uint32\_t dev, uint32\_t gridId, CUDBGGridStatus \*status)

Check whether the grid corresponding to the given gridId is still present on the device.

### Parameters

**dev**

**gridId**

- grid ID

**status**

- enum indicating whether the grid status is INVALID, PENDING, ACTIVE, SLEEPING, TERMINATED or UNDETERMINED

### Returns

CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_GRID,  
CUDBG\_ERROR\_INTERNAL

Since CUDA 5.0.

Deprecated in CUDA 5.5.

## CUDBGResult (\*CUDBGAPI\_st::getTID) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*tid)

Get the ID of the Linux thread hosting the context of the grid.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**tid**

- the returned thread id

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_GRID,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

## 3.8. Device Properties

**CUDBGResult (\*CUDBGAPI\_st::getDeviceName)**  
**(uint32\_t dev, char \*buf, uint32\_t sz)**

Get the device name string.

### Parameters

**dev**

- device index

**buf**

- the destination buffer

**sz**

- the size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_BUFFER\_TOO\_SMALL,  
CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 6.5.

See also:

[getSMType](#)

[getDeviceType](#)

**CUDBGResult (\*CUDBGAPI\_st::getDeviceType)**  
**(uint32\_t dev, char \*buf, uint32\_t sz)**

Get the string description of the device.

### Parameters

**dev**

- device index

**buf**

- the destination buffer

**sz**

- the size of the buffer

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_BUFFER\_TOO\_SMALL,  
 CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[getSMType](#)

## CUDBGResult (\*CUDBGAPI\_st::getNumDevices) (uint32\_t \*numDev)

Get the number of installed CUDA devices.

### Parameters

**numDev**

- the returned number of devices

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[getNumSMs](#)

[getNumWarps](#)

[getNumLanes](#)

[getNumRegisters](#)

## CUDBGResult (\*CUDBGAPI\_st::getNumLanes) (uint32\_t dev, uint32\_t \*numLanes)

Get the number of lanes per warp on the device.

### Parameters

**dev**

- device index

**numLanes**

- the returned number of lanes

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[getNumDevices](#)

[getNumSMs](#)

[getNumWarps](#)

[getNumRegisters](#)

## CUDBGResult (\*CUDBGAPI\_st::getNumPredicates) (uint32\_t dev, uint32\_t \*numPredicates)

Get the number of predicate registers per lane on the device.

### Parameters

#### **dev**

- device index

#### **numPredicates**

- the returned number of predicate registers

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 6.5.

See also:

[getNumDevices](#)

[getNumSMs](#)

[getNumWarps](#)

[getNumLanes](#)

[getNumRegisters](#)

## CUDBGResult (\*CUDBGAPI\_st::getNumRegisters) (uint32\_t dev, uint32\_t \*numRegs)

Get the number of registers per lane on the device.

### Parameters

**dev**

- device index

**numRegs**

- the returned number of registers

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[getNumDevices](#)

[getNumSMs](#)

[getNumWarps](#)

[getNumLanes](#)

## CUDBGResult (\*CUDBGAPI\_st::getNumSMs) (uint32\_t dev, uint32\_t \*numSMs)

Get the total number of SMs on the device.

### Parameters

**dev**

- device index

**numSMs**

- the returned number of SMs

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[getNumDevices](#)

[getNumWarps](#)

[getNumLanes](#)

[getNumRegisters](#)

## CUDBGResult

**(\*CUDBGAPI\_st::getNumUniformPredicates) (uint32\_t dev, uint32\_t \*numPredicates)**

Get the number of uniform predicate registers per warp on the device.

### Parameters

**dev**

- device index

**numPredicates**

- the returned number of uniform predicate registers

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 10.0.

See also:

[getNumUniformPredicates](#)

## CUDBGResult

**(\*CUDBGAPI\_st::getNumUniformRegisters) (uint32\_t dev, uint32\_t \*numRegs)**

Get the number of uniform registers per warp on the device.

### Parameters

**dev**

- device index

**numRegs**

- the returned number of uniform registers

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 10.0.

See also:

[getNumRegisters](#)

## CUDBGResult (\*CUDBGAPI\_st::getNumWarps) (uint32\_t dev, uint32\_t \*numWarps)

Get the number of warps per SM on the device.

### Parameters

**dev**

- device index

**numWarps**

- the returned number of warps

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[getNumDevices](#)

[getNumSMs](#)

[getNumLanes](#)

[getNumRegisters](#)

## CUDBGResult (\*CUDBGAPI\_st::getSmType) (uint32\_t dev, char \*buf, uint32\_t sz)

Get the SM type of the device.

### Parameters

**dev**

- device index

**buf**

- the destination buffer

**SZ**

- the size of the buffer

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_BUFFER\_TOO\_SMALL,  
CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 3.0.

See also:

[getDeviceType](#)

## 3.9. DWARF Utilities

**CUDBGResult (\*CUDBGAPI\_st::disassemble) (uint32\_t dev, uint64\_t addr, uint32\_t \*instSize, char \*buf, uint32\_t SZ)**

Disassemble instruction at instruction address.

**Parameters****dev**

- device index

**addr**

- instruction address

**instSize**

- instruction size (32 or 64 bits)

**buf**

- disassembled instruction buffer

**SZ**

- disassembled instruction buffer size

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_UNKNOWN

Since CUDA 3.0.

**CUDBGResult** (\*CUDBGAPI\_st::getElfImageByHandle)  
 (uint32\_t devId, uint64\_t handle, CUDBGElfImageType  
 type, void \*elfImage, uint64\_t size)

Get the relocated or non-relocated ELF image for the given handle on the given device.

#### Parameters

##### **devId**

- device index

##### **handle**

- elf image handle

##### **type**

- type of the requested ELF image

##### **elfImage**

- pointer to the ELF image

##### **size**

#### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_UNINITIALIZED

The handle is provided in the ELF Image Loaded notification event.

Since CUDA 6.0.

**CUDBGResult**  
 (\*CUDBGAPI\_st::getHostAddrFromDeviceAddr)  
 (uint32\_t dev, uint64\_t device\_addr, uint64\_t \*host\_addr)

given a device virtual address, return a corresponding system memory virtual address.

#### Parameters

##### **dev**

- device index

##### **device\_addr**

- device memory address

##### **host\_addr**

- returned system memory address

## Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE, CUDBG\_ERROR\_INVALID\_CONTEXT, CUDBG\_ERROR\_INVALID\_MEMORY\_SEGMENT

Since CUDA 4.1.

See also:

[readGenericMemory](#)

[writeGenericMemory](#)

**CUDBGResult (\*CUDBGAPI\_st::getPhysicalRegister30)**  
**(uint64\_t pc, char \*reg, uint32\_t \*buf, uint32\_t sz, uint32\_t**  
**\*numPhysRegs, CUDBGRegClass \*regClass)**

Get the physical register number(s) assigned to a virtual register name 'reg' at a given PC, if 'reg' is live at that PC.

## Parameters

### **pc**

- Program counter

### **reg**

- virtual register index

### **buf**

- physical register name(s)

### **sz**

- the physical register name buffer size

### **numPhysRegs**

- number of physical register names returned

### **regClass**

- the class of the physical registers

## Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_UNKNOWN\_FUNCTION, CUDBG\_ERROR\_UNKNOWN

Since CUDA 3.0.

[Deprecated](#) in CUDA 3.1.

**CUDBGResult** (\*CUDBGAPI\_st::getPhysicalRegister40)  
 (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t pc, char  
 \*reg, uint32\_t \*buf, uint32\_t sz, uint32\_t \*numPhysRegs,  
 CUDBGRegClass \*regClass)

Get the physical register number(s) assigned to a virtual register name 'reg' at a given PC, if 'reg' is live at that PC.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp indx

**pc**

- Program counter

**reg**

- virtual register index

**buf**

- physical register name(s)

**sz**

- the physical register name buffer size

**numPhysRegs**

- number of physical register names returned

**regClass**

- the class of the physical registers

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_ARGS,  
 CUDBG\_ERROR\_UNKNOWN\_FUNCTION, CUDBG\_ERROR\_UNKNOWN

Get the physical register number(s) assigned to a virtual register name 'reg' at a given PC, if 'reg' is live at that PC. If a virtual register name is mapped to more than one physical register, the physical register with the lowest physical register index will contain the highest bits of the virtual register, and the physical register with the highest physical register index will contain the lowest bits.

Since CUDA 3.1.

Deprecated in CUDA 4.1.

## CUDBGResult (\*CUDBGAPI\_st::isDeviceCodeAddress) (uintptr\_t addr, bool \*isDeviceAddress)

Determines whether a virtual address resides within device code.

### Parameters

#### **addr**

- virtual address

#### **isDeviceAddress**

- true if address resides within device code

### Returns

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_SUCCESS

Since CUDA 3.0.

## CUDBGResult (\*CUDBGAPI\_st::isDeviceCodeAddress55) (uintptr\_t addr, bool \*isDeviceAddress)

Determines whether a virtual address resides within device code. This API is strongly deprecated. Use CUDBGAPI\_st::isDeviceCodeAddress instead.

### Parameters

#### **addr**

- virtual address

#### **isDeviceAddress**

- true if address resides within device code

### Returns

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_SUCCESS

Since CUDA 3.0.

Deprecated in CUDA 6.0

## CUDBGResult

(\*CUDBGAPI\_st::lookupDeviceCodeSymbol) (char \*symName, bool \*symFound, uintptr\_t \*symAddr)

Determines whether a symbol represents a function in device code and returns its virtual address.

### Parameters

#### symName

- symbol name

#### symFound

- set to true if the symbol is found

#### symAddr

- the symbol virtual address if found

### Returns

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_UNINITIALIZED, CUDBG\_SUCCESS

Since CUDA 3.0.

## 3.10. Events

One of those events will create a [CUDBGEvent](#):

- ▶ the elf image of the current kernel has been loaded and the addresses within its DWARF sections have been relocated (and can now be used to set breakpoints),
- ▶ a device breakpoint has been hit,
- ▶ a CUDA kernel is ready to be launched,
- ▶ a CUDA kernel has terminated.

When a [CUDBGEvent](#) is created, the debugger is notified by calling the callback functions registered with `setNotifyNewEventCallback()` after the API struct initialization. It is up to the debugger to decide what method is best to be notified. The debugger API routines cannot be called from within the callback function or the routine will return an error.

Upon notification the debugger is responsible for handling the [CUDBGEvents](#) in the event queue by using `CUDBGAPI_st::getNextEvent()`, and for acknowledging the debugger API that the event has been handled by calling `CUDBGAPI_st::acknowledgeEvent()`. In the case of an event raised by the device itself, such as a breakpoint being hit, the event queue will be empty. It is the responsibility of the debugger to inspect the hardware any time a [CUDBGEvent](#) is received.

Example:

```
↑ CUDBGEvent event;
```

```

CUDBGResult res;
for (res = cudbgAPI->getNextEvent(&event);
     res == CUDBG_SUCCESS && event.kind != CUDBG_EVENT_INVALID;
     res = cudbgAPI->getNextEvent(&event)) {
    switch (event.kind)
    {
        case CUDBG_EVENT_ELF_IMAGE_LOADED:
            //...
            break;
        case CUDBG_EVENT_KERNEL_READY:
            //...
            break;
        case CUDBG_EVENT_KERNEL_FINISHED:
            //...
            break;
        default:
            error(...);
    }
}

```

See `cuda-tdep.c` and `cuda-linux-nat.c` files in the `cuda-gdb` source code for a more detailed example on how to use `CUDBGEvent`.

## struct CUDBGEvent

Event information container.

## struct CUDBGEventCallbackData

Event information passed to callback set with `setNotifyNewEventCallback` function.

## struct CUDBGEventCallbackData40

Event information passed to callback set with `setNotifyNewEventCallback` function.

## enum CUDBGEventKind

CUDA Kernel Events.

### Values

**CUDBG\_EVENT\_INVALID = 0x000**

Invalid event.

**CUDBG\_EVENT\_ELF\_IMAGE\_LOADED = 0x001**

The ELF image for a CUDA source module is available.

**CUDBG\_EVENT\_KERNEL\_READY = 0x002**

A CUDA kernel is about to be launched.

**CUDBG\_EVENT\_KERNEL\_FINISHED = 0x003**

A CUDA kernel has terminated.

**CUDBG\_EVENT\_INTERNAL\_ERROR = 0x004**

An internal error occur. The debugging framework may be unstable.

**CUDBG\_EVENT\_CTX\_PUSH = 0x005**

A CUDA context was pushed.

**CUDBG\_EVENT\_CTX\_POP = 0x006**

A CUDA CTX was popped.

**CUDBG\_EVENT\_CTX\_CREATE = 0x007**

A CUDA CTX was created.

**CUDBG\_EVENT\_CTX\_DESTROY = 0x008**

A CUDA context was destroyed.

**CUDBG\_EVENT\_TIMEOUT = 0x009**

An timeout event is sent at regular interval. This event can safely be ignored.

**CUDBG\_EVENT\_ATTACH\_COMPLETE = 0x00a**

The attach process has completed and debugging of device code may start.

**CUDBG\_EVENT\_DETACH\_COMPLETE = 0x00b**

**CUDBG\_EVENT\_ELF\_IMAGE\_UNLOADED = 0x00c**

**CUDBG\_EVENT\_FUNCTIONS\_LOADED = 0x00d**

**CUDBG\_EVENT\_ALL\_DEVICES\_SUSPENDED = 0x00e**

**typedef (\*CUDBGNotifyNewEventCallback)  
(CUDBGEventData\* data)**

function type of the function called to notify debugger of the presence of a new event in the event queue.

**typedef (\*CUDBGNotifyNewEventCallback31) (void\*  
data)**

function type of the function called to notify debugger of the presence of a new event in the event queue.

Deprecated in CUDA 3.2.

**CUDBGResult (\*CUDBGAPI\_st::acknowledgeEvent30)  
(CUDBGEvent30 \*event)**

Inform the debugger API that the event has been processed.

### Parameters

#### event

- pointer to the event that has been processed

### Returns

CUDBG\_SUCCESS

Since CUDA 3.0.

Deprecated in CUDA 3.1.

## CUDBGResult (\*CUDBGAPI\_st::acknowledgeEvents42) ( )

Inform the debugger API that synchronous events have been processed.

### Returns

CUDBG\_SUCCESS

Since CUDA 3.1.

Deprecated in CUDA 5.0.

## CUDBGResult (\*CUDBGAPI\_st::acknowledgeSyncEvents) ( )

Inform the debugger API that synchronous events have been processed.

### Returns

CUDBG\_SUCCESS

Since CUDA 5.0.

## CUDBGResult (\*CUDBGAPI\_st::getErrorStringEx) (char \*buf, uint32\_t bufSz, uint32\_t \*msgSz)

Fills a user-provided buffer with an error message encoded as a null-terminated ASCII string. The error message is specific to the last failed API call and is invalidated after every API call.

### Parameters

#### buf

- the destination buffer

#### bufSz

- the size of the destination buffer in bytes

#### msgSz

- the size of an error message including the terminating null character.

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_BUFFER\_TOO\_SMALL

CUDBG\_ERROR\_INVALID\_ARGS, CUDBG\_ERROR\_UNINITIALIZED

Since CUDA 12.2.

See also:

getErrorString

## CUDBGResult (\*CUDBGAPI\_st::getNextAsyncEvent50) (CUDBGEvent50 \*event)

Copies the next available event in the asynchronous event queue into 'event' and removes it from the queue. The asynchronous event queue is held separate from the normal event queue, and does not require acknowledgement from the debug client.

### Parameters

#### event

- pointer to an event container where to copy the event parameters

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_INVALID\_ARGS

Since CUDA 5.0.

Deprecated in CUDA 5.5.

## CUDBGResult (\*CUDBGAPI\_st::getNextAsyncEvent55) (CUDBGEvent55 \*event)

Copies the next available event in the asynchronous event queue into 'event' and removes it from the queue. The asynchronous event queue is held separate from the normal event queue, and does not require acknowledgement from the debug client.

### Parameters

#### event

- pointer to an event container where to copy the event parameters

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_INVALID\_ARGS

Since CUDA 5.5.

## CUDBGResult (\*CUDBGAPI\_st::getNextEvent) (CUDBGEventType type, CUDBGEvent \*event)

Copies the next available event into 'event' and removes it from the queue.

### Parameters

#### type

- application event queue type

#### event

- pointer to an event container where to copy the event parameters

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_INVALID\_ARGS

Since CUDA 6.0.

## CUDBGResult (\*CUDBGAPI\_st::getNextEvent30) (CUDBGEvent30 \*event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

### Parameters

#### event

- pointer to an event container where to copy the event parameters

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_INVALID\_ARGS

Since CUDA 3.0.

Deprecated in CUDA 3.1.

## CUDBGResult (\*CUDBGAPI\_st::getNextEvent32) (CUDBGEvent32 \*event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

### Parameters

#### event

- pointer to an event container where to copy the event parameters

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_INVALID\_ARGS

Since CUDA 3.1.

Deprecated in CUDA 4.0

## CUDBGResult (\*CUDBGAPI\_st::getNextEvent42) (CUDBGEvent42 \*event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

### Parameters

**event**

- pointer to an event container where to copy the event parameters

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_INVALID\_ARGS

Since CUDA 4.0.

Deprecated in CUDA 5.0

## CUDBGResult (\*CUDBGAPI\_st::getNextSyncEvent50) (CUDBGEvent50 \*event)

### Parameters

**event**

- pointer to an event container where to copy the event parameters

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_INVALID\_ARGS

Since CUDA 5.0.

Deprecated in CUDA 5.5.

## CUDBGResult (\*CUDBGAPI\_st::getNextSyncEvent55) (CUDBGEvent55 \*event)

Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.

### Parameters

#### **event**

- pointer to an event container where to copy the event parameters

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_INVALID\_ARGS

Since CUDA 5.5.

## CUDBGResult (\*CUDBGAPI\_st::setNotifyNewEventCallback) (CUDBGNotifyNewEventCallback callback)

Provides the API with the function to call to notify the debugger of a new application or device event.

### Parameters

#### **callback**

- the callback function

### Returns

CUDBG\_SUCCESS

Since CUDA 4.1.

## CUDBGResult (\*CUDBGAPI\_st::setNotifyNewEventCallback31) (CUDBGNotifyNewEventCallback31 callback, void \*data)

Provides the API with the function to call to notify the debugger of a new application or device event.

### Parameters

#### **callback**

- the callback function

**data**

- a pointer to be passed to the callback when called

**Returns**

CUDBG\_SUCCESS

Since CUDA 3.0.

Deprecated in CUDA 3.2.

**CUDBGResult**

(\*CUDBGAPI\_st::setNotifyNewEventCallback40)

(CUDBGNotifyNewEventCallback40 callback)

Provides the API with the function to call to notify the debugger of a new application or device event.

**Parameters****callback**

- the callback function

**Returns**

CUDBG\_SUCCESS

Since CUDA 3.2.

Deprecated in CUDA 4.1.

## 3.3. Debugger API

The API reference guide for the CUDA debugger.

---

# Chapter 4. Data Structures

Here are the data structures with brief descriptions:

## **cudaGetAPI**

The CUDA debugger API routines

## **CUDBGEvent**

Event information container

### **CUDBGEvent::CUDBGEvent::cases st**

### **CUDBGEvent::CUDBGEvent::cases st::CUDBGEvent::cases st::contextCreate st**

### **CUDBGEvent::CUDBGEvent::cases st::CUDBGEvent::cases st::contextDestroy st**

### **CUDBGEvent::CUDBGEvent::cases st::CUDBGEvent::cases st::contextPop st**

### **CUDBGEvent::CUDBGEvent::cases st::CUDBGEvent::cases st::contextPush st**

### **CUDBGEvent::CUDBGEvent::cases st::CUDBGEvent::cases st::elfImageLoaded st**

### **CUDBGEvent::CUDBGEvent::cases st::CUDBGEvent::cases st::internalError st**

### **CUDBGEvent::CUDBGEvent::cases st::CUDBGEvent::cases st::kernelFinished st**

### **CUDBGEvent::CUDBGEvent::cases st::CUDBGEvent::cases st::kernelReady st**

## **CUDBGEventCallbackData**

Event information passed to callback set with setNotifyNewEventCallback function

## **CUDBGEventCallbackData40**

Event information passed to callback set with setNotifyNewEventCallback function

## **CUDBGGridInfo**

Grid info

## 4.2. CUDBGEvent Struct Reference

Event information container.

## CUDBGEvent::cases

Information for each type of event.

## CUDBGEventKind CUDBGEvent::kind

Event type.

### 4.3. CUDBGEvent::cases\_st Union Reference

```
struct CUDBGEvent::cases_st::contextCreate_st
CUDBGEvent::cases_st::contextCreate
```

Information about the context being created.

```
struct CUDBGEvent::cases_st::contextDestroy_st
CUDBGEvent::cases_st::contextDestroy
```

Information about the context being destroyed.

```
struct CUDBGEvent::cases_st::contextPop_st
CUDBGEvent::cases_st::contextPop
```

Information about the context being popped.

```
struct CUDBGEvent::cases_st::contextPush_st
CUDBGEvent::cases_st::contextPush
```

Information about the context being pushed.

```
struct CUDBGEvent::cases_st::elfImageLoaded_st
CUDBGEvent::cases_st::elfImageLoaded
```

Information about the loaded ELF image.

```
struct CUDBGEvent::cases_st::internalError_st
CUDBGEvent::cases_st::internalError
```

Information about internal errors.

```
struct CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::cases_st::kernelFinished
```

Information about the kernel that just terminated.

```
struct CUDBGEvent::cases_st::kernelReady_st
CUDBGEvent::cases_st::kernelReady
```

Information about the kernel ready to be launched.

## 4.4. CUDBGEvent::cases\_st::contextCreate\_st Struct Reference

`uint64_t CUDBGEvent::cases_st::contextCreate_st::context`  
the context being created.

`uint32_t CUDBGEvent::cases_st::contextCreate_st::dev`  
device index of the context.

`uint32_t CUDBGEvent::cases_st::contextCreate_st::tid`  
host thread id (or LWP id) of the thread hosting the context (Linux only).

## 4.5. CUDBGEvent::cases\_st::contextDestroy\_st Struct Reference

`uint64_t CUDBGEvent::cases_st::contextDestroy_st::context`  
the context being destroyed.

`uint32_t CUDBGEvent::cases_st::contextDestroy_st::dev`  
device index of the context.

`uint32_t CUDBGEvent::cases_st::contextDestroy_st::tid`  
host thread id (or LWP id) of the thread hosting the context (Linux only).

## 4.6. CUDBGEvent::cases\_st::contextPop\_st Struct Reference

`uint64_t CUDBGEvent::cases_st::contextPop_st::context`  
the context being popped.

`uint32_t CUDBGEvent::cases_st::contextPop_st::dev`  
device index of the context.

`uint32_t CUDBGEvent::cases_st::contextPop_st::tid`  
host thread id (or LWP id) of the thread hosting the context (Linux only).

## 4.7. CUDBGEvent::cases\_st::contextPush\_st Struct Reference

`uint64_t CUDBGEvent::cases_st::contextPush_st::context`  
the context being pushed.

`uint32_t CUDBGEvent::cases_st::contextPush_st::dev`  
device index of the context.

`uint32_t CUDBGEvent::cases_st::contextPush_st::tid`  
host thread id (or LWP id) of the thread hosting the context (Linux only).

## 4.8. CUDBGEvent::cases\_st::elfImageLoaded\_st Struct Reference

`uint64_t`

`CUDBGEvent::cases_st::elfImageLoaded_st::context`

context of the kernel.

`uint32_t CUDBGEvent::cases_st::elfImageLoaded_st::dev`

device index of the kernel.

`uint64_t`

`CUDBGEvent::cases_st::elfImageLoaded_st::handle`

ELF image handle.

`uint64_t`

`CUDBGEvent::cases_st::elfImageLoaded_st::module`

module of the kernel.

`uint32_t`

`CUDBGEvent::cases_st::elfImageLoaded_st::properties`

ELF image properties.

`uint64_t CUDBGEvent::cases_st::elfImageLoaded_st::size`

size of the ELF image (64-bit).

## 4.9. `CUDBGEvent::cases_st::internalError_st` Struct Reference

`CUDBGResult`

`CUDBGEvent::cases_st::internalError_st::errorType`

Type of the internal error.

## 4.10. `CUDBGEvent::cases_st::kernelFinished_st` Struct Reference

uint64\_t

CUDBGEvent::cases\_st::kernelFinished\_st::context

context of the kernel.

uint32\_t CUDBGEvent::cases\_st::kernelFinished\_st::dev

device index of the kernel.

uint64\_t

CUDBGEvent::cases\_st::kernelFinished\_st::function

function of the kernel.

uint64\_t

CUDBGEvent::cases\_st::kernelFinished\_st::functionEntry

entry PC of the kernel.

uint64\_t CUDBGEvent::cases\_st::kernelFinished\_st::gridId

grid index of the kernel.

uint64\_t

CUDBGEvent::cases\_st::kernelFinished\_st::module

module of the kernel.

uint32\_t CUDBGEvent::cases\_st::kernelFinished\_st::tid

host thread id (or LWP id) of the thread hosting the kernel (Linux only).

## 4.11. CUDBGEvent::cases\_st::kernelReady\_st Struct Reference

## CuDim3

### `CUDBGEvent::cases_st::kernelReady_st::blockDim`

block dimensions of the kernel.

### `uint64_t CUDBGEvent::cases_st::kernelReady_st::context`

context of the kernel.

### `uint32_t CUDBGEvent::cases_st::kernelReady_st::dev`

device index of the kernel.

### `uint64_t CUDBGEvent::cases_st::kernelReady_st::function`

function of the kernel.

## `uint64_t`

### `CUDBGEvent::cases_st::kernelReady_st::functionEntry`

entry PC of the kernel.

## CuDim3

### `CUDBGEvent::cases_st::kernelReady_st::gridDim`

grid dimensions of the kernel.

### `uint64_t CUDBGEvent::cases_st::kernelReady_st::gridId`

grid index of the kernel.

### `uint64_t CUDBGEvent::cases_st::kernelReady_st::module`

module of the kernel.

## `uint64_t`

### `CUDBGEvent::cases_st::kernelReady_st::parentGridId`

64-bit grid index of the parent grid.

### `uint32_t CUDBGEvent::cases_st::kernelReady_st::tid`

host thread id (or LWP id) of the thread hosting the kernel (Linux only).

## CUDBGKernelType

### CUDBGEvent::cases\_st::kernelReady\_st::type

the type of the kernel: system or application.

## 4.12. CUDBGEventCallbackData Struct Reference

Event information passed to callback set with setNotifyNewEventCallback function.

### uint32\_t CUDBGEventCallbackData::tid

Host thread id of the context generating the event. Zero if not available.

### uint32\_t CUDBGEventCallbackData::timeout

A boolean notifying the debugger that the debug API timed while waiting for a response from the debugger to a previous event. It is up to the debugger to decide what to do in response to a timeout.

## 4.13. CUDBGEventCallbackData40 Struct Reference

Event information passed to callback set with setNotifyNewEventCallback function.

Deprecated in CUDA 4.1.

### uint32\_t CUDBGEventCallbackData40::tid

Host thread id of the context generating the event. Zero if not available.

## 4.14. CUDBGGridInfo Struct Reference

Grid info.

## `CuDim3 CUDBGGridInfo::blockDim`

The block dimensions.

## `uint64_t CUDBGGridInfo::context`

The context this grid belongs to.

## `uint32_t CUDBGGridInfo::dev`

The index of the device this grid is running on.

## `uint64_t CUDBGGridInfo::function`

The function corresponding to this grid.

## `uint64_t CUDBGGridInfo::functionEntry`

The entry address of the function corresponding to this grid.

## `CuDim3 CUDBGGridInfo::gridDim`

The grid dimensions.

## `uint64_t CUDBGGridInfo::gridId64`

The 64-bit grid ID of this grid.

## `uint64_t CUDBGGridInfo::module`

The module this grid belongs to.

## `CUDBGKernelOrigin CUDBGGridInfo::origin`

The origin of this grid, CPU or GPU.

## `uint64_t CUDBGGridInfo::parentGridId`

The 64-bit grid ID that launched this grid.

## `uint32_t CUDBGGridInfo::tid`

The host thread ID that launched this grid.

## `CUDBGKernelType CUDBGGridInfo::type`

The type of the grid.

## 4.1. Debugger API

The API reference guide for the CUDA debugger.

---

# Chapter 5. Data Fields

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

## A

### **acknowledgeEvent30**

[cudbgGetAPI](#)

### **acknowledgeEvents42**

[cudbgGetAPI](#)

### **acknowledgeSyncEvents**

[cudbgGetAPI](#)

## B

### **blockDim**

[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)

[CUDBGGridInfo](#)

## C

### **cases**

[CUDBGEvent](#)

### **clearAttachState**

[cudbgGetAPI](#)

### **context**

[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)

[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextCreate\\_st](#)

[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextDestroy\\_st](#)

[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelFinished\\_st](#)

[CUDBGGridInfo](#)

[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::elfImageLoaded\\_st](#)

[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextPop\\_st](#)

[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextPush\\_st](#)

### **contextCreate**

[CUDBGEvent::CUDBGEvent::cases\\_st](#)

**contextDestroy**[CUDBGEvent::CUDBGEvent::cases\\_st](#)**contextPop**[CUDBGEvent::CUDBGEvent::cases\\_st](#)**contextPush**[CUDBGEvent::CUDBGEvent::cases\\_st](#)**D****dev**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::elfImageLoaded\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextPush\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextDestroy\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextCreate\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextPop\\_st](#)[CUDBGGridInfo](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelFinished\\_st](#)**disassemble**[cudbgGetAPI](#)**E****elfImageLoaded**[CUDBGEvent::CUDBGEvent::cases\\_st](#)**errorType**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::internalError\\_st](#)**F****finalize**[cudbgGetAPI](#)**function**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)[CUDBGGridInfo](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelFinished\\_st](#)**functionEntry**[CUDBGGridInfo](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelFinished\\_st](#)**G****generateCoredump**[cudbgGetAPI](#)**getAdjustedCodeAddress**[cudbgGetAPI](#)

**getBlockDim**[cudbgGetAPI](#)**getClusterDim**[cudbgGetAPI](#)**getConstBankAddress**[cudbgGetAPI](#)**getConstBankAddress123**[cudbgGetAPI](#)**getDeviceName**[cudbgGetAPI](#)**getDevicePCIBusInfo**[cudbgGetAPI](#)**getDeviceType**[cudbgGetAPI](#)**getElfImage**[cudbgGetAPI](#)**getElfImage32**[cudbgGetAPI](#)**getElfImageByHandle**[cudbgGetAPI](#)**getErrorStringEx**[cudbgGetAPI](#)**getGridAttribute**[cudbgGetAPI](#)**getGridAttributes**[cudbgGetAPI](#)**getGridDim**[cudbgGetAPI](#)**getGridDim32**[cudbgGetAPI](#)**getGridInfo**[cudbgGetAPI](#)**getGridInfo55**[cudbgGetAPI](#)**getGridStatus**[cudbgGetAPI](#)**getGridStatus50**[cudbgGetAPI](#)**getHostAddrFromDeviceAddr**[cudbgGetAPI](#)**getLoadedFunctionInfo**[cudbgGetAPI](#)

**getLoadedFunctionInfo118**

[cudbgGetAPI](#)

**getManagedMemoryRegionInfo**

[cudbgGetAPI](#)

**getNextAsyncEvent50**

[cudbgGetAPI](#)

**getNextAsyncEvent55**

[cudbgGetAPI](#)

**getNextEvent**

[cudbgGetAPI](#)

**getNextEvent30**

[cudbgGetAPI](#)

**getNextEvent32**

[cudbgGetAPI](#)

**getNextEvent42**

[cudbgGetAPI](#)

**getNextSyncEvent50**

[cudbgGetAPI](#)

**getNextSyncEvent55**

[cudbgGetAPI](#)

**getNumDevices**

[cudbgGetAPI](#)

**getNumLanes**

[cudbgGetAPI](#)

**getNumPredicates**

[cudbgGetAPI](#)

**getNumRegisters**

[cudbgGetAPI](#)

**getNumSMs**

[cudbgGetAPI](#)

**getNumUniformPredicates**

[cudbgGetAPI](#)

**getNumUniformRegisters**

[cudbgGetAPI](#)

**getNumWarps**

[cudbgGetAPI](#)

**getPhysicalRegister30**

[cudbgGetAPI](#)

**getPhysicalRegister40**

[cudbgGetAPI](#)

**getSmType**

[cudbgGetAPI](#)

**getSupportedDebuggerCapabilities**[cudbgGetAPI](#)**getTID**[cudbgGetAPI](#)**gridDim**[CUDBGGridInfo](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)**gridId**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelFinished\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)**gridId64**[CUDBGGridInfo](#)**H****handle**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::elfImageLoaded\\_st](#)**I****initialize**[cudbgGetAPI](#)**initializeAttachStub**[cudbgGetAPI](#)**internalError**[CUDBGEvent::CUDBGEvent::cases\\_st](#)**isDeviceCodeAddress**[cudbgGetAPI](#)**isDeviceCodeAddress55**[cudbgGetAPI](#)**K****kernelFinished**[CUDBGEvent::CUDBGEvent::cases\\_st](#)**kernelReady**[CUDBGEvent::CUDBGEvent::cases\\_st](#)**kind**[CUDBGEvent](#)**L****lookupDeviceCodeSymbol**[cudbgGetAPI](#)

## M

**memcheckReadErrorAddress**[cudaDbgGetAPI](#)**module**[CUDBGGridInfo](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelFinished\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::elfImageLoaded\\_st](#)

## O

**origin**[CUDBGGridInfo](#)

## P

**parentGridId**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)[CUDBGGridInfo](#)**properties**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::elfImageLoaded\\_st](#)

## R

**readActiveLanes**[cudaDbgGetAPI](#)**readAllVirtualReturnAddresses**[cudaDbgGetAPI](#)**readBlockIdx**[cudaDbgGetAPI](#)**readBlockIdx32**[cudaDbgGetAPI](#)**readBrokenWarps**[cudaDbgGetAPI](#)**readCallDepth**[cudaDbgGetAPI](#)**readCallDepth32**[cudaDbgGetAPI](#)**readCCRegister**[cudaDbgGetAPI](#)**readClusterIdx**[cudaDbgGetAPI](#)**readCodeMemory**[cudaDbgGetAPI](#)

**readConstMemory**  
[cudbgGetAPI](#)

**readDeviceExceptionState**  
[cudbgGetAPI](#)

**readDeviceExceptionState80**  
[cudbgGetAPI](#)

**readErrorPC**  
[cudbgGetAPI](#)

**readGenericMemory**  
[cudbgGetAPI](#)

**readGlobalMemory**  
[cudbgGetAPI](#)

**readGlobalMemory31**  
[cudbgGetAPI](#)

**readGlobalMemory55**  
[cudbgGetAPI](#)

**readGridId**  
[cudbgGetAPI](#)

**readGridId50**  
[cudbgGetAPI](#)

**readLaneException**  
[cudbgGetAPI](#)

**readLaneStatus**  
[cudbgGetAPI](#)

**readLocalMemory**  
[cudbgGetAPI](#)

**readParamMemory**  
[cudbgGetAPI](#)

**readPC**  
[cudbgGetAPI](#)

**readPinnedMemory**  
[cudbgGetAPI](#)

**readPredicates**  
[cudbgGetAPI](#)

**readRegister**  
[cudbgGetAPI](#)

**readRegisterRange**  
[cudbgGetAPI](#)

**readReturnAddress**  
[cudbgGetAPI](#)

**readReturnAddress32**  
[cudbgGetAPI](#)

**readSharedMemory**

[cudbgGetAPI](#)

**readSmException**

[cudbgGetAPI](#)

**readSyscallCallDepth**

[cudbgGetAPI](#)

**readTextureMemory**

[cudbgGetAPI](#)

**readTextureMemoryBindless**

[cudbgGetAPI](#)

**readThreadIdx**

[cudbgGetAPI](#)

**readUniformPredicates**

[cudbgGetAPI](#)

**readUniformRegisterRange**

[cudbgGetAPI](#)

**readValidLanes**

[cudbgGetAPI](#)

**readValidWarps**

[cudbgGetAPI](#)

**readVirtualPC**

[cudbgGetAPI](#)

**readVirtualReturnAddress**

[cudbgGetAPI](#)

**readVirtualReturnAddress32**

[cudbgGetAPI](#)

**readWarpState**

[cudbgGetAPI](#)

**readWarpState60**

[cudbgGetAPI](#)

**requestCleanupOnDetach**

[cudbgGetAPI](#)

**requestCleanupOnDetach55**

[cudbgGetAPI](#)

**resumeDevice**

[cudbgGetAPI](#)

**resumeWarpsUntilPC**

[cudbgGetAPI](#)

## S

**setBreakpoint**

[cudbgGetAPI](#)

**setBreakpoint31**[cudbgGetAPI](#)**setKernelLaunchNotificationMode**[cudbgGetAPI](#)**setNotifyNewEventCallback**[cudbgGetAPI](#)**setNotifyNewEventCallback31**[cudbgGetAPI](#)**setNotifyNewEventCallback40**[cudbgGetAPI](#)**singleStepWarp**[cudbgGetAPI](#)**singleStepWarp40**[cudbgGetAPI](#)**singleStepWarp41**[cudbgGetAPI](#)**size**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::elfImageLoaded\\_st](#)**suspendDevice**[cudbgGetAPI](#)**T****tid**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelFinished\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextPop\\_st](#)[CUDBGEventCallbackData](#)[CUDBGGridInfo](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextCreate\\_st](#)[CUDBGEventCallbackData40](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextDestroy\\_st](#)[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::contextPush\\_st](#)**timeout**[CUDBGEventCallbackData](#)**type**[CUDBGEvent::CUDBGEvent::cases\\_st::CUDBGEvent::cases\\_st::kernelReady\\_st](#)[CUDBGGridInfo](#)**U****unsetBreakpoint**[cudbgGetAPI](#)**unsetBreakpoint31**[cudbgGetAPI](#)

## W

**writeCCRegister**[cudaBgGetAPI](#)**writeGenericMemory**[cudaBgGetAPI](#)**writeGlobalMemory**[cudaBgGetAPI](#)**writeGlobalMemory31**[cudaBgGetAPI](#)**writeGlobalMemory55**[cudaBgGetAPI](#)**writeLocalMemory**[cudaBgGetAPI](#)**writeParamMemory**[cudaBgGetAPI](#)**writePinnedMemory**[cudaBgGetAPI](#)**writePredicates**[cudaBgGetAPI](#)**writeRegister**[cudaBgGetAPI](#)**writeSharedMemory**[cudaBgGetAPI](#)**writeUniformPredicates**[cudaBgGetAPI](#)**writeUniformRegister**[cudaBgGetAPI](#)

---

## Chapter 6.      Deprecated List

**Global CUDBGAPI\_st::requestCleanupOnDetach55 )(void)**

in CUDA 6.0

**Class CUDBGEventCallbackData40**

in CUDA 4.1.

**Global CUDBGAPI\_st::singleStepWarp40 )(uint32\_t dev, uint32\_t sm, uint32\_t wp)**

in CUDA 4.1.

**Global CUDBGAPI\_st::singleStepWarp41 )(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t \*warpMask)**

in CUDA 7.5.

**Global CUDBGAPI\_st::setBreakpoint31 )(uint64\_t addr)**

in CUDA 3.2.

**Global CUDBGAPI\_st::unsetBreakpoint31 )(uint64\_t addr)**

in CUDA 3.2.

**Global CUDBGAPI\_st::readBlockIdx32 )(uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim2 \*blockIdx)**

in CUDA 4.0.

**Global CUDBGAPI\_st::readCallDepth32** )(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*depth)

in CUDA 4.0.

**Global CUDBGAPI\_st::readGlobalMemory31** )(uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)

in CUDA 3.2.

**Global CUDBGAPI\_st::readGlobalMemory55** )(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint64\_t addr, void \*buf, uint32\_t sz)

in CUDA 6.0.

**Global CUDBGAPI\_st::readGridId50** )(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*gridId)

in CUDA 5.5.

**Global CUDBGAPI\_st::readReturnAddress32** )(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t level, uint64\_t \*ra)

in CUDA 4.0.

**Global CUDBGAPI\_st::readVirtualReturnAddress32** )(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t level, uint64\_t \*ra)

in CUDA 4.0.

**Global CUDBGAPI\_st::writeGlobalMemory31** )(uint32\_t dev, uint64\_t addr, const void \*buf, uint32\_t sz)

in CUDA 3.2.

**Global CUDBGAPI\_st::writeGlobalMemory55** )(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t ln, uint64\_t addr, const void \*buf, uint32\_t sz)

in CUDA 6.0.

**Global CUDBGAPI\_st::getElfImage32 )(uint32\_t dev, uint32\_t sm, uint32\_t wp, bool relocated, void \*\*elfImage, uint32\_t \*size)**

in CUDA 4.0.

**Global CUDBGAPI\_st::getGridDim32 )(uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim2 \*gridDim)**

in CUDA 4.0.

**Global CUDBGAPI\_st::getGridStatus50 )(uint32\_t dev, uint32\_t gridId, CUDBGGridStatus \*status)**

in CUDA 5.5.

**Global CUDBGAPI\_st::getPhysicalRegister30 )(uint64\_t pc, char \*reg, uint32\_t \*buf, uint32\_t sz, uint32\_t \*numPhysRegs, CUDBGRegClass \*regClass)**

in CUDA 3.1.

**Global CUDBGAPI\_st::getPhysicalRegister40 )(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t pc, char \*reg, uint32\_t \*buf, uint32\_t sz, uint32\_t \*numPhysRegs, CUDBGRegClass \*regClass)**

in CUDA 4.1.

**Global CUDBGAPI\_st::isDeviceCodeAddress55 )(uintptr\_t addr, bool \*isDeviceAddress)**

in CUDA 6.0

**Global CUDBGNotifyNewEventCallback31**

in CUDA 3.2.

**Global CUDBGAPI\_st::acknowledgeEvent30 )(CUDBGEvent30 \*event)**

in CUDA 3.1.

**Global CUDBGAPI\_st::acknowledgeEvents42 )(void)**

in CUDA 5.0.

**Global CUDBGAPI\_st::getNextAsyncEvent50 )(CUDBGEvent50 \*event)**

in CUDA 5.5.

**Global CUDBGAPI\_st::getNextEvent30 )(CUDBGEvent30 \*event)**

in CUDA 3.1.

**Global CUDBGAPI\_st::getNextEvent32 )(CUDBGEvent32 \*event)**

in CUDA 4.0

**Global CUDBGAPI\_st::getNextEvent42 )(CUDBGEvent42 \*event)**

in CUDA 5.0

**Global CUDBGAPI\_st::getNextSyncEvent50 )(CUDBGEvent50 \*event)**

in CUDA 5.5.

**Global CUDBGAPI\_st::setNotifyNewEventCallback31 )(CUDBGNotifyNewEventCallback31  
callback, void \*data)**

in CUDA 3.2.

**Global CUDBGAPI\_st::setNotifyNewEventCallback40 )(CUDBGNotifyNewEventCallback40  
callback)**

in CUDA 4.1.

## Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

## OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

## Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

## Copyright

© 2007-2024 NVIDIA Corporation & affiliates. All rights reserved.