



# CUDA DEBUGGER API

TRM-06710-001 \_v5.5 for POWER8 | August 2014

## API Reference Manual



# TABLE OF CONTENTS

<b>Chapter 1. Release Notes.....</b>	<b>1</b>
1.1. 6.0 Release.....	1
<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::initialize.....	11
3.3. Device Execution Control.....	11
CUDBGAPI_st::resumeDevice.....	11
CUDBGAPI_st::resumeWarpsUntilPC.....	12
CUDBGAPI_st::singleStepWarp.....	13
CUDBGAPI_st::singleStepWarp40.....	14
CUDBGAPI_st::suspendDevice.....	14
3.4. Breakpoints.....	15
CUDBGAPI_st::getAdjustedCodeAddress.....	15
CUDBGAPI_st::setBreakpoint.....	16
CUDBGAPI_st::setBreakpoint31.....	16
CUDBGAPI_st::unsetBreakpoint.....	17
CUDBGAPI_st::unsetBreakpoint31.....	17
3.5. Device State Inspection.....	18
CUDBGAPI_st::getManagedMemoryRegionInfo.....	18
CUDBGAPI_st::memcheckReadErrorAddress.....	19
CUDBGAPI_st::readActiveLanes.....	19
CUDBGAPI_st::readBlockIdx.....	20
CUDBGAPI_st::readBlockIdx32.....	21
CUDBGAPI_st::readBrokenWarps.....	22
CUDBGAPI_st::readCallDepth.....	23
CUDBGAPI_st::readCallDepth32.....	24
CUDBGAPI_st::readCCRegister.....	24
CUDBGAPI_st::readCodeMemory.....	26
CUDBGAPI_st::readConstMemory.....	27
CUDBGAPI_st::readErrorPC.....	28
CUDBGAPI_st::readGenericMemory.....	28

CUDBGAPI_st::readGlobalMemory.....	30
CUDBGAPI_st::readGlobalMemory31.....	31
CUDBGAPI_st::readGlobalMemory55.....	32
CUDBGAPI_st::readGridId.....	33
CUDBGAPI_st::readGridId50.....	34
CUDBGAPI_st::readLaneException.....	35
CUDBGAPI_st::readLaneStatus.....	35
CUDBGAPI_st::readLocalMemory.....	36
CUDBGAPI_st::readParamMemory.....	37
CUDBGAPI_st::readPC.....	38
CUDBGAPI_st::readPinnedMemory.....	39
CUDBGAPI_st::readPredicates.....	40
CUDBGAPI_st::readRegister.....	41
CUDBGAPI_st::readRegisterRange.....	42
CUDBGAPI_st::readReturnAddress.....	43
CUDBGAPI_st::readReturnAddress32.....	44
CUDBGAPI_st::readSharedMemory.....	45
CUDBGAPI_st::readSyscallCallDepth.....	46
CUDBGAPI_st::readTextureMemory.....	47
CUDBGAPI_st::readTextureMemoryBindless.....	48
CUDBGAPI_st::readThreadId.....	49
CUDBGAPI_st::readValidLanes.....	50
CUDBGAPI_st::readValidWarps.....	51
CUDBGAPI_st::readVirtualPC.....	52
CUDBGAPI_st::readVirtualReturnAddress.....	52
CUDBGAPI_st::readVirtualReturnAddress32.....	53
CUDBGAPI_st::readWarpState.....	54
CUDBGAPI_st::writePinnedMemory.....	55
CUDBGAPI_st::writePredicates.....	56
3.6. Device State Alteration.....	57
CUDBGAPI_st::writeCCRegister.....	57
CUDBGAPI_st::writeGenericMemory.....	58
CUDBGAPI_st::writeGlobalMemory.....	59
CUDBGAPI_st::writeGlobalMemory31.....	60
CUDBGAPI_st::writeGlobalMemory55.....	61
CUDBGAPI_st::writeLocalMemory.....	62
CUDBGAPI_st::writeParamMemory.....	63
CUDBGAPI_st::writeRegister.....	64
CUDBGAPI_st::writeSharedMemory.....	65
3.7. Grid Properties.....	65
CUDBGGridInfo.....	66
CUDBGGridStatus.....	66
CUDBGAPI_st::getBlockDim.....	66

CUDBGAPI_st::getElfImage.....	67
CUDBGAPI_st::getElfImage32.....	68
CUDBGAPI_st::getGridAttribute.....	68
CUDBGAPI_st::getGridAttributes.....	69
CUDBGAPI_st::getGridDim.....	70
CUDBGAPI_st::getGridDim32.....	70
CUDBGAPI_st::getGridInfo.....	71
CUDBGAPI_st::getGridStatus.....	72
CUDBGAPI_st::getGridStatus50.....	72
CUDBGAPI_st::getTID.....	73
3.8. Device Properties.....	73
CUDBGAPI_st::getDeviceName.....	73
CUDBGAPI_st::getDeviceType.....	74
CUDBGAPI_st::getNumDevices.....	75
CUDBGAPI_st::getNumLanes.....	75
CUDBGAPI_st::getNumPredicates.....	76
CUDBGAPI_st::getNumRegisters.....	77
CUDBGAPI_st::getNumSMs.....	77
CUDBGAPI_st::getNumWarps.....	78
CUDBGAPI_st::getSmType.....	79
3.9. DWARF Utilities.....	79
CUDBGAPI_st::disassemble.....	79
CUDBGAPI_st::getElfImageByHandle.....	80
CUDBGAPI_st::getHostAddrFromDeviceAddr.....	81
CUDBGAPI_st::getPhysicalRegister30.....	81
CUDBGAPI_st::getPhysicalRegister40.....	82
CUDBGAPI_st::isDeviceCodeAddress.....	83
CUDBGAPI_st::isDeviceCodeAddress55.....	84
CUDBGAPI_st::lookupDeviceCodeSymbol.....	84
3.10. Events.....	85
CUDBGEvent.....	86
CUDBGEventCallbackData.....	86
CUDBGEventCallbackData40.....	86
CUDBGEventKind.....	86
CUDBGNotifyNewEventCallback.....	87
CUDBGNotifyNewEventCallback31.....	87
CUDBGAPI_st::acknowledgeEvent30.....	87
CUDBGAPI_st::acknowledgeEvents42.....	87
CUDBGAPI_st::acknowledgeSyncEvents.....	88
CUDBGAPI_st::getNextAsyncEvent50.....	88
CUDBGAPI_st::getNextAsyncEvent55.....	88
CUDBGAPI_st::getNextEvent.....	89
CUDBGAPI_st::getNextEvent30.....	89

CUDBGAPI_st::getNextEvent32.....	90
CUDBGAPI_st::getNextEvent42.....	90
CUDBGAPI_st::getNextSyncEvent50.....	91
CUDBGAPI_st::getNextSyncEvent55.....	91
CUDBGAPI_st::setNotifyNewEventCallback.....	92
CUDBGAPI_st::setNotifyNewEventCallback31.....	92
CUDBGAPI_st::setNotifyNewEventCallback40.....	93
<b>Chapter 4. Data Structures.....</b>	<b>94</b>
CUDBGAPI_st.....	94
acknowledgeEvent30.....	95
acknowledgeEvents42.....	95
acknowledgeSyncEvents.....	95
clearAttachState.....	96
disassemble.....	96
finalize.....	96
getAdjustedCodeAddress.....	97
getBlockDim.....	98
getDeviceName.....	98
getDevicePCIBusInfo.....	99
getDeviceType.....	99
getElfImage.....	100
getElfImage32.....	101
getElfImageByHandle.....	101
getGridAttribute.....	102
getGridAttributes.....	103
getGridDim.....	103
getGridDim32.....	104
getGridInfo.....	105
getGridStatus.....	105
getGridStatus50.....	106
getHostAddrFromDeviceAddr.....	106
getManagedMemoryRegionInfo.....	107
getNextAsyncEvent50.....	108
getNextAsyncEvent55.....	108
getNextEvent.....	109
getNextEvent30.....	109
getNextEvent32.....	110
getNextEvent42.....	110
getNextSyncEvent50.....	111
getNextSyncEvent55.....	111
getNumDevices.....	112
getNumLanes.....	112
getNumPredicates.....	113

getNumRegisters.....	114
getNumSMs.....	114
getNumWarps.....	115
getPhysicalRegister30.....	116
getPhysicalRegister40.....	116
getSmType.....	118
getTID.....	118
initialize.....	119
initializeAttachStub.....	119
isDeviceCodeAddress.....	119
isDeviceCodeAddress55.....	120
lookupDeviceCodeSymbol.....	120
memcheckReadErrorAddress.....	121
readActiveLanes.....	122
readBlockIdx.....	123
readBlockIdx32.....	124
readBrokenWarps.....	125
readCallDepth.....	125
readCallDepth32.....	126
readCCRegister.....	127
readCodeMemory.....	128
readConstMemory.....	129
readDeviceExceptionState.....	130
readErrorPC.....	130
readGenericMemory.....	131
readGlobalMemory.....	132
readGlobalMemory31.....	133
readGlobalMemory55.....	134
readGridId.....	135
readGridId50.....	136
readLaneException.....	137
readLaneStatus.....	137
readLocalMemory.....	138
readParamMemory.....	139
readPC.....	140
readPinnedMemory.....	141
readPredicates.....	142
readRegister.....	143
readRegisterRange.....	144
readReturnAddress.....	145
readReturnAddress32.....	146
readSharedMemory.....	147
readSyscallCallDepth.....	148

readTextureMemory.....	149
readTextureMemoryBindless.....	150
readThreadId.....	151
readValidLanes.....	152
readValidWarps.....	153
readVirtualPC.....	154
readVirtualReturnAddress.....	154
readVirtualReturnAddress32.....	155
readWarpState.....	156
requestCleanupOnDetach.....	157
requestCleanupOnDetach55.....	157
resumeDevice.....	157
resumeWarpsUntilPC.....	158
setBreakpoint.....	159
setBreakpoint31.....	159
setKernelLaunchNotificationMode.....	160
setNotifyNewEventCallback.....	160
setNotifyNewEventCallback31.....	161
setNotifyNewEventCallback40.....	161
singleStepWarp.....	162
singleStepWarp40.....	162
suspendDevice.....	163
unsetBreakpoint.....	164
unsetBreakpoint31.....	164
writeCCRegister.....	165
writeGenericMemory.....	166
writeGlobalMemory.....	167
writeGlobalMemory31.....	167
writeGlobalMemory55.....	168
writeLocalMemory.....	169
writeParamMemory.....	170
writePinnedMemory.....	171
writePredicates.....	172
writeRegister.....	173
writeSharedMemory.....	174
CUDBGEvent.....	175
cases.....	175
kind.....	175
CUDBGEvent::cases_st.....	175
contextCreate.....	176
contextDestroy.....	176
contextPop.....	176
contextPush.....	176

elfImageLoaded.....	176
internalError.....	176
kernelFinished.....	176
kernelReady.....	176
CUDBGEvent::cases_st::contextCreate_st.....	176
context.....	177
dev.....	177
tid.....	177
CUDBGEvent::cases_st::contextDestroy_st.....	177
context.....	177
dev.....	177
tid.....	177
CUDBGEvent::cases_st::contextPop_st.....	177
context.....	178
dev.....	178
tid.....	178
CUDBGEvent::cases_st::contextPush_st.....	178
context.....	178
dev.....	178
tid.....	178
CUDBGEvent::cases_st::elfImageLoaded_st.....	178
context.....	179
dev.....	179
handle.....	179
module.....	179
properties.....	179
size.....	179
CUDBGEvent::cases_st::internalError_st.....	179
errorType.....	179
CUDBGEvent::cases_st::kernelFinished_st.....	179
context.....	180
dev.....	180
function.....	180
functionEntry.....	180
gridId.....	180
module.....	180
tid.....	180
CUDBGEvent::cases_st::kernelReady_st.....	180
blockDim.....	181
context.....	181
dev.....	181
function.....	181
functionEntry.....	181



gridDim.....	181
gridId.....	181
module.....	181
parentGridId.....	181
tid.....	181
type.....	182
CUDBGEventCallbackData.....	182
tid.....	182
timeout.....	182
CUDBGEventCallbackData40.....	182
tid.....	182
CUDBGGridInfo.....	182
blockDim.....	183
context.....	183
dev.....	183
function.....	183
functionEntry.....	183
gridDim.....	183
gridId64.....	183
module.....	183
origin.....	183
parentGridId.....	183
tid.....	183
type.....	183
<b>Chapter 5. Data Fields.....</b>	<b>184</b>
<b>Chapter 6. File List.....</b>	<b>193</b>
cudadebugger.h.....	193
CUDBGAPI_st.....	210
CUDBGEvent.....	210
CUDBGEventCallbackData.....	210
CUDBGEventCallbackData40.....	210
CUDBGGridInfo.....	210
<b>Chapter 8. Deprecated List.....</b>	<b>222</b>



# Chapter 1.

## RELEASE NOTES

### 1.1. 6.0 Release

#### **New optimized routines**

Following API calls were added to read bulk information about the device and speed up debugging. `ReadWarpState()` reads the whole state of a warp in a single API call. `ReadRegisterRange()` reads the value of a range of N registers in a single API call. `ResumeWarpsUntilPC()` resumes a set of warps until a given PC instead of single-stepping several times.

#### **Adjusted Code Address**

On some architectures, some PCs may be invalid and should not be referenced. To help the debugger clients, the API now provides the routine `getAdjustedCodeAddress()`. Given a code address, the function returns the corresponding valid PC.

#### **Precise Error Reporting**

Exceptions are not always precise. The device may stop at a PC other than the address of the instruction that triggered an exception. On some device architectures, it is sometimes possible to recover the address of that instruction. That address can now be retrieved using the newly introduced `readErrorPC()` API routine.

#### **ELF Image Notification Events**

When the ELF image is unloaded from the device, the debugger client is now notified with a new `CUDBG_EVENT_ELF_IMAGE_UNLOADED` event type. The ELF image load/unload events also include a new properties field that is currently only used to indicate whether an ELF image corresponds to a set of system kernels or not, which may need to be hidden from the user. Also, the ELF image events now include a handle to the actual copy of the ELF image instead of including the ELF image itself. To retrieve the ELF image, use the `getElfImageByHandle()` routine.

#### **Unified Memory Support**

Existing routines were modified to support Unified Memory and should be used instead of the old ones: `read/writeGenericMemory()` and `read/writeGlobalMemory()`. `GetManagedMemoryRegionInfo` was added to identify the address segments that are considered managed memory and that should therefore require special attention when accessed.

**Cleanup on Detach**

The detach procedure was simplified and is now symmetrical with the attach procedure. The debugger client must now check if the application needs to be resumed to complete the detach process, just as it is done for the attach process.

**State examination on a running device**

The state collection functions in the debug API will return `CUDBG_ERROR_RUNNING_DEVICE` if called without first calling `suspendDevice` to ensure the device is stopped.

**Using `singleStepWarp()` in an application using CUDA Dynamic Parallelism**

Due to changes in the way CUDA Dynamic Parallelism operates, the debug API's `singleStepWarp()` entry point can now return `CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE`. To correctly handle such cases, the debugger client must set a breakpoint at the return address of the current frame and must resume all devices and resume all host threads. When `singleStepWarp()` returns `CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE`, there is no guarantee that hardware state has not been modified. In particular, when running with software preemption, there is no guarantee that any GPU state is valid across the `singleStepWarp()` call. As a result, debugger clients must invalidate and reanalyze all GPU state after the call if `singleStepWarp()` returns an error.

**Miscellaneous**

New error values were added to support the newly added API routines. The `getNextSync/AsyncEvent()` routines were merged into a single `getNextEvent()` routine with an extra parameter instead.

# 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.

## 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 (ptxParamStorage). 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 function call to the function `cudbgApiInit()` with an argument of "2", i.e., "`cudbgApiInit(2)`". 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.
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 function call to the function `cudbgApiDetach()` in the memory space of the application. 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\_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 can not 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**

## 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

### Description

Since CUDA 3.0.

### See also:

[initialize](#)

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

Initialize the API.

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNKNOWN

### Description

Since CUDA 3.0.

### See also:

[finalize](#)

## 3.3. Device Execution Control

## CUDBGResult (\*CUDBGAPI\_st::resumeDevice) (uint32\_t dev)

Resume a suspended CUDA device.

### Parameters

**dev**

- device index

**Returns**

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

**Description**

Since CUDA 3.0.

**See also:**

[suspendDevice](#)

[singleStepWarp](#)

## CUDBGResult (\*CUDBGAPI\_st::resumeWarpsUntilPC) (uint32\_t devId, uint32\_t sm, uint64\_t warpMask, uint64\_t virtPC)

Inserts a temporary breakpoint at the specified virtual PC, and resumes all warps in the specified bitmask on a given SM. As compared to CUDBGAPI\_st::resumeDevice, CUDBGAPI\_st::resumeWarpsUntilPC provides finer-grain control by resuming a selected set of warps on the same SM. The main intended usage is to accelerate the single-stepping process when the target PC is known in advance. Instead of single-stepping each warp individually until the target PC is hit, the client can issue this API. When this API is used, errors within CUDA kernels will no longer be reported precisely. In the situation where resuming warps is not possible, this API will return CUDBG\_ERROR\_WARP\_RESUME\_NOT\_POSSIBLE. The client should then fall back to using CUDBGAPI\_st::singleStepWarp or CUDBGAPI\_st::resumeDevice.

**Parameters****devId**

- device index

**sm**

- the SM index

**warpMask**

- the bitmask of warps to resume (1 = resume, 0 = do not resume)

**virtPC**

- the virtual PC where the temporary breakpoint will be inserted

**Returns**

CUDBG\_SUCCESS CUDBG\_ERROR\_INVALID\_ARGS  
CUDBG\_ERROR\_INVALID\_DEVICE CUDBG\_ERROR\_INVALID\_SM  
CUDBG\_ERROR\_INVALID\_WARP\_MASK

CUDBG\_ERROR\_WARP\_RESUME\_NOT\_POSSIBLE  
CUDBG\_ERROR\_UNINITIALIZED

### Description

Since CUDA 6.0.

### See also:

[resumeDevice](#)

**CUDBGResult (\*CUDBGAPI\_st::singleStepWarp) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t \*warpMask)**

Single step an individual warp on a suspended CUDA device.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**warpMask**

- the warps that have been single-stepped

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_RUNNING\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_UNKNOWN CUDBG\_ERROR\_WARP\_RESUME\_NOT\_POSSIBLE

### Description

Since CUDA 4.1.

### See also:

[resumeDevice](#)

[suspendDevice](#)

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

Single step an individual warp on a suspended CUDA device.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_RUNNING\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_UNKNOWN CUDBG\_ERROR\_WARP\_RESUME\_NOT\_POSSIBLE

### Description

Since CUDA 3.0.

Deprecated in CUDA 4.1.

### See also:

[resumeDevice](#)

[suspendDevice](#)

[singleStepWarp](#)

## CUDBGResult (\*CUDBGAPI\_st::suspendDevice)(uint32\_t dev)

Suspends a running CUDA device.

### Parameters

**dev**

- device index



**Returns**

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

**Description**

Since CUDA 3.0.

**See also:**

[resumeDevice](#)

[singleStepWarp](#)

## 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

**Description**

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

### Description

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

### Description

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

### Description

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

**Description**

Since CUDA 3.0.

Deprecated in CUDA 3.2.

**See also:**

[setBreakpoint31](#)

## 3.5. Device State Inspection

### 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

**Description**

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

### Description

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

**Description**

Since CUDA 3.0.

**See also:**[readGridId](#)[readBlockIdx](#)[readThreadId](#)[readBrokenWarps](#)[readValidWarps](#)[readValidLanes](#)

**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

**Description**

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

**Description**

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

### Description

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

#### Description

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

#### Description

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

## Description

Since CUDA 6.5.

## See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readGlobalMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

[readPredicates](#)

## 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

### Description

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

### Description

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

#### Description

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

**Description**

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\_

### Description

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

### Description

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

### Description

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

### Description

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

### Description

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

#### Description

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

**Description**

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

## Description

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

**Description**

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

**Description**

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

**Description**

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

### Description

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

#### Description

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

**Description**

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

### Description

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

### Description

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

**Description**

Since CUDA 3.0.

**See also:**[readCodeMemory](#)[readConstMemory](#)[readGenericMemory](#)[readParamMemory](#)

[readLocalMemory](#)

[readTextureMemory](#)

[readRegister](#)

[readPC](#)

**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

### Description

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)

Read the content of texture memory with given id and coords on sm\_20 and lower.

### 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\_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

### Description

Read the content of texture memory with given id and coords on sm\_20 and lower.

On sm\_30 and higher, use [CUDBGAPI\\_st::readTextureMemoryBindless](#) instead.

Since CUDA 4.0.

### 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)`

Read the content of texture memory with given symtab index and coords on sm\_30 and higher.

### 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\_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

**Description**

Read the content of texture memory with given symtab index and coords on sm\_30 and higher.

For sm\_20 and lower, use `CUDBGAPI_st::readTextureMemory` instead.

Since CUDA 4.2.

**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

### Description

Since CUDA 3.0.

### See also:

[readGridId](#)

[readBlockIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

**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

### Description

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

**Description**

Since CUDA 3.0.

**See also:**[readGridId](#)[readBlockIdx](#)[readThreadIdx](#)[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

#### Description

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

**Description**

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

**Description**

Since CUDA 3.1.

[Deprecated](#) in CUDA 4.0.

**See also:**

[readCallDepth32](#)

[readReturnAddress32](#)

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



**Description**

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

**Description**

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

### Description

Since CUDA 6.5.

### See also:

[writeConstMemory](#)

[writeGenericMemory](#)

[writeGlobalMemory](#)

[writeParamMemory](#)

[writeSharedMemory](#)

[writeTextureMemory](#)

[writeLocalMemory](#)

[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

### Description

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

### Description

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\_

### Description

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

#### Description

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

### Description

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

#### Description

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

#### Description

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

#### Description

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

#### Description

Since CUDA 3.0.

#### See also:

[writeGenericMemory](#)

[writeParamMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

## 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

**Description**

Since CUDA 3.0.

**See also:**

[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

**Description**

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

#### Description

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

**Description**

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

**Description**

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

### Description

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

**Description**

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

**Description**

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

### Description

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

### Description

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

#### Description

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

**Description**

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

**Description**

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

### Description

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

**Description**

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

**Description**

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

### Description

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

**Description**

Since CUDA 3.0.

**See also:**

[getNumDevices](#)

[getNumWarps](#)

[getNumLanes](#)

[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

**Description**

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

### Description

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

**Description**

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

**Description**

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

### Description

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

**Description**

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

**Description**

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

**Description**

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

### Description

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

### Description

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 ge 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**



**typedef (\*CUDBGNotifyNewEventCallback)  
(CUDBGEventCallbackData\* 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

#### Description

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

#### Description

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

### Description

Since CUDA 5.0.

## 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

### Description

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

**Description**

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

**Description**

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

**Description**

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

**Description**

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

**Description**

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

### Description

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

### Description

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

### Description

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

### Description

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

### Description

Since CUDA 3.2.

[Deprecated](#) in CUDA 4.1.

# Chapter 4.

## DATA STRUCTURES

Here are the data structures with brief descriptions:

### **cudbgGetAPI**

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

## 4.1. CUDBGAPI\_st Struct Reference

The CUDA debugger API routines.



## CUDBGResult (\*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

### Description

Since CUDA 3.0.

Deprecated in CUDA 3.1.

## CUDBGResult (\*acknowledgeEvents42) ()

Inform the debugger API that synchronous events have been processed.

### Returns

CUDBG\_SUCCESS

### Description

Since CUDA 3.1.

Deprecated in CUDA 5.0.

## CUDBGResult (\*acknowledgeSyncEvents) ()

Inform the debugger API that synchronous events have been processed.

### Returns

CUDBG\_SUCCESS

### Description

Since CUDA 5.0.

## CUDBGResult (\*clearAttachState) ()

Clear attach-specific state prior to detach.

### Returns

CUDBG\_SUCCESS

### Description

Since CUDA 5.0.

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

## CUDBGResult (\*finalize) ()

Finalize the API and free all memory.

### Returns

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

**Description**

Since CUDA 3.0.

**See also:**

[initialize](#)

## **CUDBGResult (\*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

**Description**

Since CUDA 5.5.

**See also:**

[unsetBreakpoint](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[getGridDim](#)

## CUDBGResult (\*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

**Description**

Since CUDA 6.5.

**See also:**

`getSMType`

`getDeviceType`

**CUDBGResult (\*getDevicePCIBusInfo) (uint32\_t devId,  
 uint32\_t \*pciBusId, uint32\_t \*pciDevId)**

Get PCI bus and device ids associated with device devId.

**Parameters****devId**

- the cuda device id

**pciBusId**

- pointer where corresponding PCI BUS ID would be stored

**pciDevId**

- pointer where corresponding PCI DEVICE ID would be stored

**Returns**

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

**CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**

getSMType

**CUDBGResult (\*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

**Description**

Since CUDA 4.0.

**CUDBGResult (\*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

#### Description

Since CUDA 3.0.

[Deprecated](#) in CUDA 4.0.

**CUDBGResult (\*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

**Description**

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

Since CUDA 6.0.

**CUDBGResult (\*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

**Description**

Since CUDA 3.1.



**CUDBGResult (\*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

#### Description

Since CUDA 3.1.

**CUDBGResult (\*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

**Description**

Since CUDA 4.0.

**See also:**

[getBlockDim](#)

## CUDBGResult (\*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

**Description**

Since CUDA 3.0.

[Deprecated](#) in CUDA 4.0.

**See also:**

[getBlockDim](#)

## CUDBGResult (\*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

### Description

Since CUDA 5.5.

## CUDBGResult (\*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

### Description

Since CUDA 5.5.

## CUDBGResult (\*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

### Description

Since CUDA 5.0.

Deprecated in CUDA 5.5.

## CUDBGResult (\*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

**Description**

Since CUDA 4.1.

**See also:**

[readGenericMemory](#)

[writeGenericMemory](#)

## **CUDBGResult (\*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

**Description**

Since CUDA 6.0.

## CUDBGResult (\*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

### Description

Since CUDA 5.0.

Deprecated in CUDA 5.5.

## CUDBGResult (\*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

### Description

Since CUDA 5.5.

## CUDBGResult (\*getNextEvent) (CUDBGEventQueueType 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

### Description

Since CUDA 6.0.

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

Deprecated in CUDA 3.1.

## CUDBGResult (\*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

### Description

Since CUDA 3.1.

Deprecated in CUDA 4.0

## CUDBGResult (\*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

### Description

Since CUDA 4.0.

Deprecated in CUDA 5.0



## CUDBGResult (\*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

### Description

Since CUDA 5.0.

Deprecated in CUDA 5.5.

## CUDBGResult (\*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

### Description

Since CUDA 5.5.

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[getNumSMs](#)

[getNumWarps](#)

[getNumLanes](#)

[getNumRegisters](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

**See also:**[getNumDevices](#)[getNumSMs](#)[getNumWarps](#)[getNumRegisters](#)

## CUDBGResult (\*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

**Description**

Since CUDA 6.5.

**See also:**[getNumDevices](#)[getNumSMs](#)[getNumWarps](#)[getNumLanes](#)[getNumRegisters](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[getNumDevices](#)

[getNumSMs](#)

[getNumWarps](#)

[getNumLanes](#)

## CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**

[getNumDevices](#)

[getNumWarps](#)

[getNumLanes](#)

[getNumRegisters](#)

## **CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**

[getNumDevices](#)

[getNumSMs](#)

[getNumLanes](#)

[getNumRegisters](#)

**CUDBGResult (\*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

#### Description

Since CUDA 3.0.

Deprecated in CUDA 3.1.

**CUDBGResult (\*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 index

**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

### Description

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 (\*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

### Description

Since CUDA 3.0.

See also:

[getDeviceType](#)

## CUDBGResult (\*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

**Description**

Since CUDA 3.0.

## CUDBGResult (\*initialize) ()

Initialize the API.

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNKNOWN

**Description**

Since CUDA 3.0.

**See also:**

[finalize](#)

## CUDBGResult (\*initializeAttachStub) ()

Initialize the attach stub.

**Returns**

CUDBG\_SUCCESS

**Description**

Since CUDA 5.0.

## CUDBGResult (\*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

**Description**

Since CUDA 3.0.

## CUDBGResult (\*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

**Description**

Since CUDA 3.0.

Deprecated in CUDA 6.0

## CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**CUDBGResult (\*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\_INVALID\_ARGS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_LANE, CUDBG\_ERROR\_INVALID\_SM,  
CUDBG\_ERROR\_INVALID\_WARP, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_MEMCHECK\_NOT\_ENABLED, CUDBG\_SUCCESS

**Description**

Since CUDA 5.0.

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[readGridId](#)

[readBlockIdx](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

## CUDBGResult (\*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

### Description

Since CUDA 4.0.

### See also:

[readGridId](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

Deprecated in CUDA 4.0.

### See also:

[readGridId](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[readGridId](#)

[readBlockIdx](#)

[readThreadIdx](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*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

**Description**

Since CUDA 4.0.

**See also:**

[readReturnAddress](#)

[readVirtualReturnAddress](#)

## CUDBGResult (\*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



**Description**

Since CUDA 3.1.

Deprecated in CUDA 4.0.

**See also:**

[readReturnAddress32](#)

[readVirtualReturnAddress32](#)

## **CUDBGResult (\*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

**Description**

Since CUDA 6.5.

**See also:**

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readGlobalMemory](#)  
[readParamMemory](#)  
[readSharedMemory](#)  
[readTextureMemory](#)  
[readLocalMemory](#)  
[readRegister](#)  
[readPC](#)  
[readPredicates](#)

## **CUDBGResult (\*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

### **Description**

Since CUDA 3.0.

### **See also:**

[readConstMemory](#)  
[readGenericMemory](#)  
[readParamMemory](#)  
[readSharedMemory](#)

[readTextureMemory](#)[readLocalMemory](#)[readRegister](#)[readPC](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[readCodeMemory](#)[readGenericMemory](#)[readParamMemory](#)[readSharedMemory](#)[readTextureMemory](#)[readLocalMemory](#)[readRegister](#)[readPC](#)

## CUDBGResult (\*readDeviceExceptionState) (uint32\_t devId, uint64\_t \*exceptionSMMask)

Get the exception state of the SMs on the device.

### Parameters

#### devId

- the cuda device id

#### exceptionSMMask

- Bit field containing a 1 at  $(1 \ll i)$  if SM  $i$  hit an exception

### Returns

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

## CUDBGResult (\*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

### Description

Since CUDA 6.0

**CUDBGResult (\*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

### Description

Since CUDA 6.0.

### See also:

[readCodeMemory](#)

[readConstMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)[readRegister](#)[readPC](#)

## CUDBGResult (\*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\_

### Description

Since CUDA 6.0.

### See also:

[readCodeMemory](#)[readConstMemory](#)[readParamMemory](#)[readSharedMemory](#)[readTextureMemory](#)[readLocalMemory](#)[readRegister](#)[readPC](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

Deprecated in CUDA 3.2.

### See also:

[readCodeMemory](#)

[readConstMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*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

### Description

Since CUDA 3.2.

Deprecated in CUDA 6.0.

### See also:

[readCodeMemory](#)

[readConstMemory](#)

[readParamMemory](#)

[readSharedMemory](#)



[readTextureMemory](#)[readLocalMemory](#)[readRegister](#)[readPC](#)

## **CUDBGResult (\*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

### **Description**

Since CUDA 5.5.

### **See also:**

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

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

Deprecated in CUDA 5.5.

### See also:

[readBlockIdx](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.1.

## CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readSharedMemory](#)

[readTextureMemory](#)

[readLocalMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**

[readCodeMemory](#)  
[readConstMemory](#)  
[readGenericMemory](#)  
[readParamMemory](#)  
[readSharedMemory](#)  
[readTextureMemory](#)  
[readLocalMemory](#)  
[readRegister](#)  
[readVirtualPC](#)

## **CUDBGResult (\*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

**Description**

Since CUDA 3.2.

**See also:**

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

[readSharedMemory](#)[readTextureMemory](#)[readLocalMemory](#)[readRegister](#)[readPC](#)

**CUDBGResult (\*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

### Description

Since CUDA 6.5.

### See also:

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



`readGenericMemory``readGlobalMemory``readParamMemory``readSharedMemory``readTextureMemory``readLocalMemory``readRegister``readPC`

**CUDBGResult (\*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

#### Description

Since CUDA 3.0.

#### See also:

[readCodeMemory](#)  
[readConstMemory](#)  
[readGenericMemory](#)  
[readParamMemory](#)  
[readSharedMemory](#)  
[readTextureMemory](#)  
[readLocalMemory](#)  
[readPC](#)

**CUDBGResult (\*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

#### Description

Since CUDA 6.0.

**See also:**

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

**CUDBGResult (\*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

**Description**

Since CUDA 4.0.

**See also:**

[readCallDepth](#)

[readVirtualReturnAddress](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.1.

[Deprecated](#) in CUDA 4.0.

**See also:**

[readCallDepth32](#)

[readVirtualReturnAddress32](#)

**CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

[readParamMemory](#)

[readLocalMemory](#)

[readTextureMemory](#)

[readRegister](#)

[readPC](#)

**CUDBGResult (\*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

#### Description

Since CUDA 4.1.

#### See also:

[readReturnAddress](#)

[readVirtualReturnAddress](#)

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

Read the content of texture memory with given id and coords on sm\_20 and lower.

### 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\_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

### Description

Read the content of texture memory with given id and coords on sm\_20 and lower.

On sm\_30 and higher, use [CUDBGAPI\\_st::readTextureMemoryBindless](#) instead.

Since CUDA 4.0.

### See also:

[readCodeMemory](#)

[readConstMemory](#)

[readGenericMemory](#)

`readParamMemory``readSharedMemory``readLocalMemory``readRegister``readPC`

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

Read the content of texture memory with given symtab index and coords on sm\_30 and higher.

#### 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\_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



**Description**

Read the content of texture memory with given symtab index and coords on sm\_30 and higher.

For sm\_20 and lower, use `CUDBGAPI_st::readTextureMemory` instead.

Since CUDA 4.2.

**See also:**

`readCodeMemory`

`readConstMemory`

`readGenericMemory`

`readParamMemory`

`readSharedMemory`

`readLocalMemory`

`readRegister`

`readPC`

**`CUDBGResult (*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

### Description

Since CUDA 3.0.

### See also:

[readGridId](#)

[readBlockIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[readGridId](#)

[readBlockIdx](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidWarps](#)

[readActiveLanes](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[readGridId](#)

[readBlockIdx](#)

[readThreadIdx](#)

[readBrokenWarps](#)

[readValidLanes](#)

[readActiveLanes](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

### See also:

[readPC](#)

## CUDBGResult (\*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

**Description**

Since CUDA 4.0.

**See also:**

[readCallDepth](#)

[readReturnAddress](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.1.

Deprecated in CUDA 4.0.

**See also:**

[readCallDepth32](#)

[readReturnAddress32](#)

## CUDBGResult (\*readWarpState) (uint32\_t devId, uint32\_t sm, uint32\_t wp, CUDBGWarpState \*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,

**Description**

Since CUDA 6.0.

## CUDBGResult (\*requestCleanupOnDetach) (uint32\_t appResumeFlag)

Request for cleanup of driver state when detaching.

### Parameters

#### appResumeFlag

- value of CUDBG\_RESUME\_FOR\_ATTACH\_DETACH as read from the application's process space.

### Returns

CUDBG\_SUCCESS CUDBG\_ERROR\_COMMUNICATION\_FAILURE  
CUDBG\_ERROR\_INVALID\_ARGS CUDBG\_ERROR\_INTERNAL

### Description

Since CUDA 6.0.

## CUDBGResult (\*requestCleanupOnDetach55) ()

Request for cleanup of driver state when detaching.

### Returns

CUDBG\_SUCCESS CUDBG\_ERROR\_COMMUNICATION\_FAILURE  
CUDBG\_ERROR\_INVALID\_ARGS CUDBG\_ERROR\_INTERNAL

### Description

Since CUDA 5.0.

Deprecated in CUDA 6.0

## CUDBGResult (\*resumeDevice) (uint32\_t dev)

Resume a suspended CUDA device.

### Parameters

#### dev

- device index

### Returns

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

**Description**

Since CUDA 3.0.

**See also:**

[suspendDevice](#)

[singleStepWarp](#)

## CUDBGResult (\*resumeWarpsUntilPC) (uint32\_t devId, uint32\_t sm, uint64\_t warpMask, uint64\_t virtPC)

Inserts a temporary breakpoint at the specified virtual PC, and resumes all warps in the specified bitmask on a given SM. As compared to `CUDBGAPI_st::resumeDevice`, `CUDBGAPI_st::resumeWarpsUntilPC` provides finer-grain control by resuming a selected set of warps on the same SM. The main intended usage is to accelerate the single-stepping process when the target PC is known in advance. Instead of single-stepping each warp individually until the target PC is hit, the client can issue this API. When this API is used, errors within CUDA kernels will no longer be reported precisely. In the situation where resuming warps is not possible, this API will return `CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE`. The client should then fall back to using `CUDBGAPI_st::singleStepWarp` or `CUDBGAPI_st::resumeDevice`.

**Parameters****devId**

- device index

**sm**

- the SM index

**warpMask**

- the bitmask of warps to resume (1 = resume, 0 = do not resume)

**virtPC**

- the virtual PC where the temporary breakpoint will be inserted

**Returns**

CUDBG\_SUCCESS CUDBG\_ERROR\_INVALID\_ARGS  
 CUDBG\_ERROR\_INVALID\_DEVICE CUDBG\_ERROR\_INVALID\_SM  
 CUDBG\_ERROR\_INVALID\_WARP\_MASK  
 CUDBG\_ERROR\_WARP\_RESUME\_NOT\_POSSIBLE  
 CUDBG\_ERROR\_UNINITIALIZED

**Description**

Since CUDA 6.0.



**See also:**

[resumeDevice](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.2.

**See also:**

[unsetBreakpoint](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

Deprecated in CUDA 3.2.

See also:

[unsetBreakpoint31](#)

## CUDBGResult (\*setKernelLaunchNotificationMode) (CUDBGKernelLaunchNotifyMode mode)

Set the launch notification policy.

### Parameters

#### mode

- mode to deliver kernel launch notifications in

### Returns

CUDBG\_SUCCESS

### Description

Since CUDA 5.5.

## CUDBGResult (\*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

### Description

Since CUDA 4.1.

## CUDBGResult (\*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

### Description

Since CUDA 3.0.

Deprecated in CUDA 3.2.

## CUDBGResult (\*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

### Description

Since CUDA 3.2.

Deprecated in CUDA 4.1.

## CUDBGResult (\*singleStepWarp) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint64\_t \*warpMask)

Single step an individual warp on a suspended CUDA device.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**warpMask**

- the warps that have been single-stepped

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_RUNNING\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_UNKNOWN CUDBG\_ERROR\_WARP\_RESUME\_NOT\_POSSIBLE

### Description

Since CUDA 4.1.

### See also:

[resumeDevice](#)

[suspendDevice](#)

## CUDBGResult (\*singleStepWarp40) (uint32\_t dev, uint32\_t sm, uint32\_t wp)

Single step an individual warp on a suspended CUDA device.

### Parameters

**dev**

- device index

**sm**

- SM index

**wp**

- warp index

**Returns**

CUDBG\_SUCCESS, CUDBG\_ERROR\_INVALID\_DEVICE,  
 CUDBG\_ERROR\_INVALID\_SM, CUDBG\_ERROR\_INVALID\_WARP,  
 CUDBG\_ERROR\_RUNNING\_DEVICE, CUDBG\_ERROR\_UNINITIALIZED,  
 CUDBG\_ERROR\_UNKNOWN CUDBG\_ERROR\_WARP\_RESUME\_NOT\_POSSIBLE

**Description**

Since CUDA 3.0.

Deprecated in CUDA 4.1.

**See also:**

[resumeDevice](#)

[suspendDevice](#)

[singleStepWarp](#)

## CUDBGResult (\*suspendDevice) (uint32\_t dev)

Suspends a running CUDA device.

**Parameters**

**dev**

- device index

**Returns**

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

**Description**

Since CUDA 3.0.

**See also:**

[resumeDevice](#)

[singleStepWarp](#)

## CUDBGResult (\*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

### Description

Since CUDA 3.2.

See also:

[setBreakpoint](#)

## CUDBGResult (\*unsetBreakpoint31) (uint64\_t addr)

Unsets a breakpoint at the given instruction address.

### Parameters

**addr**

- instruction address

### Returns

CUDBG\_SUCCESS, CUDBG\_ERROR\_UNINITIALIZED

### Description

Since CUDA 3.0.

[Deprecated](#) in CUDA 3.2.

See also:

[setBreakpoint31](#)

## CUDBGResult (\*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

### Description

Since CUDA 6.5.

### See also:

[writeConstMemory](#)

[writeGenericMemory](#)

[writeGlobalMemory](#)

[writeParamMemory](#)

[writeSharedMemory](#)

[writeTextureMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

[writePredicates](#)

**CUDBGResult (\*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

### Description

Since CUDA 6.0.

### See also:

[writeParamMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)



## CUDBGResult (\*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\_

### Description

Since CUDA 6.0.

### See also:

[writeParamMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

## CUDBGResult (\*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

**Description**

Since CUDA 3.0.

[Deprecated](#) in CUDA 3.2.

**See also:**

[writeParamMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.2.

Deprecated in CUDA 6.0.

**See also:**

[writeParamMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**

[writeGenericMemory](#)

[writeParamMemory](#)

[writeSharedMemory](#)

[writeRegister](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**

[writeGenericMemory](#)

[writeSharedMemory](#)

[writeLocalMemory](#)

[writeRegister](#)

## CUDBGResult (\*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

**Description**

Since CUDA 3.2.

**See also:**

[readCodeMemory](#)  
[readConstMemory](#)  
[readGenericMemory](#)  
[readParamMemory](#)  
[readSharedMemory](#)  
[readLocalMemory](#)  
[readRegister](#)  
[readPC](#)

**CUDBGResult (\*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

#### Description

Since CUDA 6.5.

**See also:**[writeConstMemory](#)[writeGenericMemory](#)[writeGlobalMemory](#)[writeParamMemory](#)[writeSharedMemory](#)[writeTextureMemory](#)[writeLocalMemory](#)[writeRegister](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**[writeGenericMemory](#)[writeParamMemory](#)[writeSharedMemory](#)[writeLocalMemory](#)

**CUDBGResult (\*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

**Description**

Since CUDA 3.0.

**See also:**[writeGenericMemory](#)[writeParamMemory](#)



`writeLocalMemory`

`writeRegister`

## 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.

# 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****getAdjustedCodeAddress**

cudbgGetAPI

**getBlockDim**  
    cudbgGetAPI  
**getDeviceName**  
    cudbgGetAPI  
**getDevicePCIBusInfo**  
    cudbgGetAPI  
**getDeviceType**  
    cudbgGetAPI  
**getElfImage**  
    cudbgGetAPI  
**getElfImage32**  
    cudbgGetAPI  
**getElfImageByHandle**  
    cudbgGetAPI  
**getGridAttribute**  
    cudbgGetAPI  
**getGridAttributes**  
    cudbgGetAPI  
**getGridDim**  
    cudbgGetAPI  
**getGridDim32**  
    cudbgGetAPI  
**getGridInfo**  
    cudbgGetAPI  
**getGridStatus**  
    cudbgGetAPI  
**getGridStatus50**  
    cudbgGetAPI  
**getHostAddrFromDeviceAddr**  
    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
getNumWarps
    cudbgGetAPI
getPhysicalRegister30
    cudbgGetAPI
getPhysicalRegister40
    cudbgGetAPI
getSmType
    cudbgGetAPI
getTID
    cudbgGetAPI
gridDim
    CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
    CUDBGGridInfo
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**

`cudbgGetAPI`

**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**

cudbgGetAPI

**readBlockIdx**

cudbgGetAPI

**readBlockIdx32**

cudbgGetAPI

**readBrokenWarps**

cudbgGetAPI

**readCallDepth**

cudbgGetAPI

**readCallDepth32**

cudbgGetAPI

**readCCRegister**

cudbgGetAPI

**readCodeMemory**

cudbgGetAPI

**readConstMemory**

cudbgGetAPI

**readDeviceExceptionState**

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  
**readSyscallCallDepth**  
    cudbgGetAPI  
**readTextureMemory**  
    cudbgGetAPI  
**readTextureMemoryBindless**  
    cudbgGetAPI  
**readThreadIdX**  
    cudbgGetAPI  
**readValidLanes**  
    cudbgGetAPI  
**readValidWarps**  
    cudbgGetAPI  
**readVirtualPC**  
    cudbgGetAPI  
**readVirtualReturnAddress**  
    cudbgGetAPI  
**readVirtualReturnAddress32**  
    cudbgGetAPI  
**readWarpState**  
    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

**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**[cudbgGetAPI](#)**writeGenericMemory**[cudbgGetAPI](#)**writeGlobalMemory**[cudbgGetAPI](#)**writeGlobalMemory31**[cudbgGetAPI](#)**writeGlobalMemory55**[cudbgGetAPI](#)**writeLocalMemory**[cudbgGetAPI](#)**writeParamMemory**[cudbgGetAPI](#)**writePinnedMemory**[cudbgGetAPI](#)**writePredicates**[cudbgGetAPI](#)**writeRegister**[cudbgGetAPI](#)**writeSharedMemory**[cudbgGetAPI](#)

# Chapter 6.

## FILE LIST

Here is a list of all documented files with brief descriptions:

### **`cuda debugger.h`**

Header file for the CUDA debugger API

## 6.1. `cuda debugger.h`

Header file for the CUDA debugger API.

### `cuda debugger.h`

```
↑/*
 * Copyright 2007-2014 NVIDIA Corporation. All rights reserved.
 *
 * NOTICE TO LICENSEE:
 *
 * This source code and/or documentation ("Licensed Deliverables") are
 * subject to NVIDIA intellectual property rights under U.S. and
 * international Copyright laws.
 *
 * These Licensed Deliverables contained herein is PROPRIETARY and
 * CONFIDENTIAL to NVIDIA and is being provided under the terms and
 * conditions of a form of NVIDIA software license agreement by and
 * between NVIDIA and Licensee ("License Agreement") or electronically
 * accepted by Licensee. Notwithstanding any terms or conditions to
 * the contrary in the License Agreement, reproduction or disclosure
 * of the Licensed Deliverables to any third party without the express
 * written consent of NVIDIA is prohibited.
 *
 * NOTWITHSTANDING ANY TERMS OR CONDITIONS TO THE CONTRARY IN THE
 * LICENSE AGREEMENT, NVIDIA MAKES NO REPRESENTATION ABOUT THE
 * SUITABILITY OF THESE LICENSED DELIVERABLES FOR ANY PURPOSE. IT IS
 * PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY OF ANY KIND.
 * NVIDIA DISCLAIMS ALL WARRANTIES WITH REGARD TO THESE LICENSED
 * DELIVERABLES, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY,
 * NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.
 * NOTWITHSTANDING ANY TERMS OR CONDITIONS TO THE CONTRARY IN THE
 * LICENSE AGREEMENT, IN NO EVENT SHALL NVIDIA BE LIABLE FOR ANY
 * SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
 * WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
 * ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
 * OF THESE LICENSED DELIVERABLES.
```

```

*
* U.S. Government End Users. These Licensed Deliverables are a
* "commercial item" as that term is defined at 48 C.F.R. 2.101 (OCT
* 1995), consisting of "commercial computer software" and "commercial
* computer software documentation" as such terms are used in 48
* C.F.R. 12.212 (SEPT 1995) and is provided to the U.S. Government
* only as a commercial end item. Consistent with 48 C.F.R.12.212 and
* 48 C.F.R. 227.7202-1 through 227.7202-4 (JUNE 1995), all
* U.S. Government End Users acquire the Licensed Deliverables with
* only those rights set forth herein.
*
* Any use of the Licensed Deliverables in individual and commercial
* software must include, in the user documentation and internal
* comments to the code, the above Disclaimer and U.S. Government End
* Users Notice.
*/

/*----- Includes
-----*/

#ifndef CUDADEBUGGER_H
#define CUDADEBUGGER_H

#include <stdlib.h>
#include "cuda_stdint.h"

#if defined(__STDC__)
#include <inttypes.h>
#include <stdbool.h>
#endif

#ifdef __cplusplus
extern "C" {
#endif

#if defined(_WIN32) && !defined(_WIN64)
/* Windows 32-bit */
#define PRIxPTR "I32x"
#endif

#if defined(_WIN64)
/* Windows 64-bit */
#define PRIxPTR "I64x"
#endif

#if defined(_WIN32)
/* Windows 32- and 64-bit */
#define PRIx64 "I64x"
#define PRId64 "I64d"
typedef unsigned char bool;
#undef false
#undef true
#define false 0
#define true 1
#endif

/*----- API Version
-----*/

#define CUDBG_API_VERSION_MAJOR 6 /* Major release version number */
#define CUDBG_API_VERSION_MINOR 5 /* Minor release version number */
#define CUDBG_API_VERSION_REVISION 121 /* Revision (build) number */

/*----- Constants
-----*/

```

```

#define CUDBG_MAX_DEVICES 32 /* Maximum number of supported devices */
#define CUDBG_MAX_SMS 64 /* Maximum number of SMS per device */
#define CUDBG_MAX_WARPS 64 /* Maximum number of warps per SM */
#define CUDBG_MAX_LANES 32 /* Maximum number of lanes per warp */

/*----- Thread/Block Coordinates Types
-----*/

typedef struct { uint32_t x, y; } CuDim2; /* DEPRECATED */
typedef struct { uint32_t x, y, z; } CuDim3; /* 3-dimensional
coordinates for threads,... */

/*----- Memory Segments (as used in DWARF)
-----*/

typedef enum {
    ptxUNSPECIFIEDStorage,
    ptxCodeStorage,
    ptxRegStorage,
    ptxSregStorage,
    ptxConstStorage,
    ptxGlobalStorage,
    ptxLocalStorage,
    ptxParamStorage,
    ptxSharedStorage,
    ptxSurfStorage,
    ptxTexStorage,
    ptxTexSamplerStorage,
    ptxGenericStorage,
    ptxIPParamStorage,
    ptxOPParamStorage,
    ptxFrameStorage,
    ptxMAXStorage
} ptxStorageKind;

/*----- Debugger System Calls
-----*/

#define CUDBG_IPC_FLAG_NAME cudbgIpcFlag
#define CUDBG_RPC_ENABLED cudbgRpcEnabled
#define CUDBG_APICLIENT_PID cudbgApiClientPid
#define CUDBG_DEBUGGER_INITIALIZED cudbgDebuggerInitialized
#define CUDBG_APICLIENT_REVISION cudbgApiClientRevision
#define CUDBG_SESSION_ID cudbgSessionId
#define CUDBG_ATTACH_HANDLER_AVAILABLE cudbgAttachHandlerAvailable
#define CUDBG_DETACH_SUSPENDED_DEVICES_MASK
cudbgDetachSuspendedDevicesMask
#define CUDBG_ENABLE_LAUNCH_BLOCKING cudbgEnableLaunchBlocking
#define CUDBG_ENABLE_INTEGRATED_MEMCHECK cudbgEnableIntegratedMemcheck
#define CUDBG_ENABLE_PREEMPTION_DEBUGGING cudbgEnablePreemptionDebugging
#define CUDBG_RESUME_FOR_ATTACH_DETACH cudbgResumeForAttachDetach

/*----- Internal Breakpoint Entries for Error Reporting
-----*/

#define CUDBG_REPORT_DRIVER_API_ERROR
cudbgReportDriverApiError
#define CUDBG_REPORT_DRIVER_API_ERROR_FLAGS
cudbgReportDriverApiErrorFlags
#define CUDBG_REPORTED_DRIVER_API_ERROR_CODE
cudbgReportedDriverApiErrorCode
#define CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_SIZE
cudbgReportedDriverApiErrorFuncNameSize
#define CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_ADDR
cudbgReportedDriverApiErrorFuncNameAddr
#define CUDBG_REPORT_DRIVER_INTERNAL_ERROR
cudbgReportDriverInternalError

```

```

#define CUDBG_REPORTED_DRIVER_INTERNAL_ERROR_CODE
cudbgReportedDriverInternalErrorCode

/*----- API Return Types
-----*/

typedef enum {
    CUDBG_SUCCESS                                = 0x0000, /* Successful
execution */
    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 (read/write) */
    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 can't 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, /* device is
suspended */
    CUDBG_ERROR_RUNNING_DEVICE                 = 0x0010, /* device is
running and not suspended */
    CUDBG_ERROR_INVALID_ADDRESS                 = 0x0012, /* address is out-
of-range */
    CUDBG_ERROR_INCOMPATIBLE_API                = 0x0013, /* API version does
not match */
    CUDBG_ERROR_INITIALIZATION_FAILURE           = 0x0014, /* The CUDA Driver
failed to initialize */
    CUDBG_ERROR_INVALID_GRID                    = 0x0015, /* Specified grid
cannot be found */
    CUDBG_ERROR_NO_EVENT_AVAILABLE              = 0x0016, /* No event left to
be processed */
    CUDBG_ERROR_SOME_DEVICES_WATCHDOGGED        = 0x0017, /* One or more
devices have an associated watchdog (eg. X) */
    CUDBG_ERROR_ALL_DEVICES_WATCHDOGGED         = 0x0018, /* All devices have
an associated watchdog (eg. X) */
    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, /* Memory is not
unmapped and can't 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, /* The
resumeWarpsUntilPC() API is not possible, use resumeDevice() or
singleStepWarp() instead */
        CUDBG_ERROR_INVALID_WARP_MASK = 0x002a, /* Specified warp
mask is zero, or contains invalid warps */
        CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS = 0x002b, /* Address cannot
be resolved to a GPU unambiguously */
        CUDBG_ERROR_RECURSIVE_API_CALL = 0x002c, /* Debug API entry
point called from within a debug API callback */
    } CUDBGResult;

    /*----- API Error Reporting Flags
-----*/
    typedef enum {
        CUDBG_REPORT_DRIVER_API_ERROR_FLAGS_NONE = 0x0000, /* Default is that
there is no flag */
        CUDBG_REPORT_DRIVER_API_ERROR_FLAGS_SUPPRESS_NOT_READY = ( 1U <<
0 ), /* When set, cudaErrorNotReady/cuErrorNotReady will not be reported */
    } CUDBGReportDriverApiErrorFlags;

    /*----- Grid Attributes
-----*/

    typedef enum {
        CUDBG_ATTR_GRID_LAUNCH_BLOCKING = 0x000, /* Whether the grid
launch is blocking or not. */
        CUDBG_ATTR_GRID_TID = 0x001, /* Id of the host thread
that launched the grid. */
    } CUDBGAttribute;

    typedef struct {
        CUDBGAttribute attribute;
        uint64_t value;
    } CUDBGAttributeValuePair;

    typedef enum {
        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 PENDING or
TERMINATED */
    } CUDBGGridStatus;

    /*----- Kernel Types
-----*/

    typedef enum {
        CUDBG_KNL_TYPE_UNKNOWN                = 0x000,    /* Any type not listed
below. */
        CUDBG_KNL_TYPE_SYSTEM                  = 0x001,    /* System kernel, such
as MemCpy. */
        CUDBG_KNL_TYPE_APPLICATION             = 0x002,    /* Application kernel,
user-defined or libraries. */
    } CUDBGKernelType;

    /*----- Elf Image Properties
-----*/

    typedef enum {
        CUDBG_ELF_IMAGE_PROPERTIES_SYSTEM     = 0x001,    /* ELF image contains
system kernels. */
    } CUDBGElfImageProperties;

    /*----- Physical Register Types
-----*/

    typedef enum {
        REG_CLASS_INVALID                     = 0x000,    /* invalid register */
        REG_CLASS_REG_CC                      = 0x001,    /* Condition register */
        REG_CLASS_REG_PRED                    = 0x002,    /* Predicate register */
        REG_CLASS_REG_ADDR                    = 0x003,    /* Address register */
        REG_CLASS_REG_HALF                    = 0x004,    /* 16-bit register
(Currently unused) */
        REG_CLASS_REG_FULL                    = 0x005,    /* 32-bit register */
        REG_CLASS_MEM_LOCAL                   = 0x006,    /* register spilled in
memory */
        REG_CLASS_LMEM_REG_OFFSET              = 0x007,    /* register at stack
offset (ABI only) */
    } CUDBGRegClass;

    /*----- Application Events
-----*/

    typedef enum {
        CUDBG_EVENT_INVALID                   = 0x000,    /* Invalid event */
        CUDBG_EVENT_ELF_IMAGE_LOADED           = 0x001,    /* ELF image for CUDA
kernel(s) is ready */
        CUDBG_EVENT_KERNEL_READY              = 0x002,    /* A CUDA kernel is
ready to be launched */
        CUDBG_EVENT_KERNEL_FINISHED           = 0x003,    /* A CUDA kernel has
terminated */
        CUDBG_EVENT_INTERNAL_ERROR             = 0x004,    /* Unexpected error. The
API may be unstable. */
        CUDBG_EVENT_CTX_PUSH                  = 0x005,    /* A CUDA context has
been pushed. */
        CUDBG_EVENT_CTX_POP                   = 0x006,    /* A CUDA context has
been popped. */
        CUDBG_EVENT_CTX_CREATE                 = 0x007,    /* A CUDA context has
been created and pushed. */
        CUDBG_EVENT_CTX_DESTROY               = 0x008,    /* A CUDA context has
been, popped if pushed, then destroyed. */
    }

```



```

    CUDBG_EVENT_TIMEOUT = 0x009, /* Nothing happened for
a while. This is heartbeat event. */
    CUDBG_EVENT_ATTACH_COMPLETE = 0x00a, /* Attach complete. */
    CUDBG_EVENT_DETACH_COMPLETE = 0x00b, /* Detach complete. */
    CUDBG_EVENT_ELF_IMAGE_UNLOADED = 0x00c, /* ELF image for CUDA
kernels(s) no longer available */
} CUDBGEventKind;

/*----- Kernel Origin
-----*/

typedef enum {
    CUDBG_KNL_ORIGIN_CPU = 0x000, /* The kernel was
launched from the CPU. */
    CUDBG_KNL_ORIGIN_GPU = 0x001, /* The kernel was
launched from the GPU. */
} CUDBGKernelOrigin;

/*----- Kernel Launch Notify Mode
-----*/

typedef enum {
    CUDBG_KNL_LAUNCH_NOTIFY_EVENT = 0x000, /* The kernel
notifications generate events */
    CUDBG_KNL_LAUNCH_NOTIFY_DEFER = 0x001, /* The kernel
notifications are deferred */
} CUDBGKernelLaunchNotifyMode;

/*----- Application Event Queue Type
-----*/

typedef enum {
    CUDBG_EVENT_QUEUE_TYPE_SYNC = 0, /* Synchronous event queue */
    CUDBG_EVENT_QUEUE_TYPE_ASYNC = 1, /* Asynchronous event queue */
} CUDBGEventQueueType;

/*----- Elf Image Type
-----*/

typedef enum {
    CUDBG_ELF_IMAGE_TYPE_NONRELOCATED = 0, /* Non-relocated ELF
image */
    CUDBG_ELF_IMAGE_TYPE_RELOCATED = 1, /* Relocated ELF image
*/
} CUDBGElfImageType;

/*----- Code Address
-----*/

typedef enum {
    CUDBG_ADJ_PREVIOUS_ADDRESS = 0x000, /* Get the adjusted
previous code address. */
    CUDBG_ADJ_CURRENT_ADDRESS = 0x001, /* Get the adjusted
current code address. */
    CUDBG_ADJ_NEXT_ADDRESS = 0x002, /* Get the adjusted next
code address. */
} CUDBGAdjAddrAction;

/* Deprecated */
typedef struct {
    CUDBGEventKind kind;
    union cases30_st {
        struct elfImageLoaded30_st {
            char *relocatedElfImage;
            char *nonRelocatedElfImage;
            uint32_t size;
        } elfImageLoaded;
    }

```

```

        struct kernelReady30_st {
            uint32_t dev;
            uint32_t gridId;
            uint32_t tid;
        } kernelReady;
        struct kernelFinished30_st {
            uint32_t dev;
            uint32_t gridId;
            uint32_t tid;
        } kernelFinished;
    } cases;
} CUDBGEvent30;

/* Deprecated */
typedef struct {
    CUDBGEventKind kind;
    union cases32_st {
        struct elfImageLoaded32_st {
            char      *relocatedElfImage;
            char      *nonRelocatedElfImage;
            uint32_t   size;
            uint32_t   dev;
            uint64_t   context;
            uint64_t   module;
        } elfImageLoaded;
        struct kernelReady32_st {
            uint32_t dev;
            uint32_t gridId;
            uint32_t tid;
            uint64_t context;
            uint64_t module;
            uint64_t function;
            uint64_t functionEntry;
        } kernelReady;
        struct kernelFinished32_st {
            uint32_t dev;
            uint32_t gridId;
            uint32_t tid;
            uint64_t context;
            uint64_t module;
            uint64_t function;
            uint64_t functionEntry;
        } kernelFinished;
        struct contextPush32_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t context;
        } contextPush;
        struct contextPop32_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t context;
        } contextPop;
        struct contextCreate32_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t context;
        } contextCreate;
        struct contextDestroy32_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t context;
        } contextDestroy;
    } cases;
} CUDBGEvent32;

/* Deprecated */

```

```

typedef struct {
    CUDBGEventKind kind;
    union cases42_st {
        struct elfImageLoaded42_st {
            char      *relocatedElfImage;
            char      *nonRelocatedElfImage;
            uint32_t  size32;
            uint32_t  dev;
            uint64_t  context;
            uint64_t  module;
            uint64_t  size;
        } elfImageLoaded;
        struct kernelReady42_st {
            uint32_t dev;
            uint32_t gridId;
            uint32_t tid;
            uint64_t context;
            uint64_t module;
            uint64_t function;
            uint64_t functionEntry;
            CuDim3   gridDim;
            CuDim3   blockDim;
            CUDBGKernelType type;
        } kernelReady;
        struct kernelFinished42_st {
            uint32_t dev;
            uint32_t gridId;
            uint32_t tid;
            uint64_t context;
            uint64_t module;
            uint64_t function;
            uint64_t functionEntry;
        } kernelFinished;
        struct contextPush42_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t context;
        } contextPush;
        struct contextPop42_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t context;
        } contextPop;
        struct contextCreate42_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t context;
        } contextCreate;
        struct contextDestroy42_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t context;
        } contextDestroy;
    } cases;
} CUDBGEvent42;

typedef struct {
    CUDBGEventKind kind;
    union cases50_st {
        struct elfImageLoaded50_st {
            char      *relocatedElfImage;
            char      *nonRelocatedElfImage;
            uint32_t  size32;
            uint32_t  dev;
            uint64_t  context;
            uint64_t  module;
            uint64_t  size;
        }
    }
}

```

```

    } elfImageLoaded;
    struct kernelReady50_st{
        uint32_t dev;
        uint32_t gridId;
        uint32_t tid;
        uint64_t context;
        uint64_t module;
        uint64_t function;
        uint64_t functionEntry;
        CuDim3   gridDim;
        CuDim3   blockDim;
        CUDBGKernelType type;
    } kernelReady;
    struct kernelFinished50_st {
        uint32_t dev;
        uint32_t gridId;
        uint32_t tid;
        uint64_t context;
        uint64_t module;
        uint64_t function;
        uint64_t functionEntry;
    } kernelFinished;
    struct contextPush50_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextPush;
    struct contextPop50_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextPop;
    struct contextCreate50_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextCreate;
    struct contextDestroy50_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextDestroy;
    struct internalError50_st {
        CUDBGResult errorType;
    } internalError;
    } cases;
} CUDBGEvent50;

typedef struct {
    CUDBGEventKind kind;
    union cases55_st {
        struct elfImageLoaded55_st {
            char      *relocatedElfImage;
            char      *nonRelocatedElfImage;
            uint32_t   size32;
            uint32_t   dev;
            uint64_t   context;
            uint64_t   module;
            uint64_t   size;
        } elfImageLoaded;
        struct kernelReady55_st{
            uint32_t dev;
            uint32_t gridId;
            uint32_t tid;
            uint64_t context;
            uint64_t module;
            uint64_t function;
        }
    }
}

```

```

        uint64_t functionEntry;
        CuDim3   gridDim;
        CuDim3   blockDim;
        CUDBGKernelType type;
        uint64_t parentGridId;
        uint64_t gridId64;
        CUDBGKernelOrigin origin;
    } kernelReady;
    struct kernelFinished55_st {
        uint32_t dev;
        uint32_t gridId;
        uint32_t tid;
        uint64_t context;
        uint64_t module;
        uint64_t function;
        uint64_t functionEntry;
        uint64_t gridId64;
    } kernelFinished;
    struct contextPush55_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextPush;
    struct contextPop55_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextPop;
    struct contextCreate55_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextCreate;
    struct contextDestroy55_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextDestroy;
    struct internalError55_st {
        CUDBGResult errorType;
    } internalError;
    } cases;
} CUDBGEvent55;

#pragma pack(push,1)
typedef struct {
    CUDBGEventKind kind;
    union cases_st {
        struct elfImageLoaded_st {
            uint32_t dev;
            uint64_t context;
            uint64_t module;
            uint64_t size;
            uint64_t handle;
            uint32_t properties;
        } elfImageLoaded;
        struct elfImageUnloaded_st {
            uint32_t dev;
            uint64_t context;
            uint64_t module;
            uint64_t size;
            uint64_t handle;
        } elfImageUnloaded;
        struct kernelReady_st {
            uint32_t dev;
            uint32_t tid;
            uint64_t gridId;

```

```

        uint64_t context;
        uint64_t module;
        uint64_t function;
        uint64_t functionEntry;
        CuDim3   gridDim;
        CuDim3   blockDim;
        CUDBGKernelType type;
        uint64_t parentGridId;
        CUDBGKernelOrigin origin;
    } kernelReady;
    struct kernelFinished_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
        uint64_t module;
        uint64_t function;
        uint64_t functionEntry;
        uint64_t gridId;
    } kernelFinished;
    struct contextPush_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextPush;
    struct contextPop_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextPop;
    struct contextCreate_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextCreate;
    struct contextDestroy_st {
        uint32_t dev;
        uint32_t tid;
        uint64_t context;
    } contextDestroy;
    struct internalError_st {
        CUDBGResult errorType;
    } internalError;
    } cases;
} CUDBGEvent;
#pragma pack(pop)

```

```

typedef struct {
    uint32_t tid;
} CUDBGEventCallbackData40;

```

```

typedef struct {
    uint32_t tid;
    uint32_t timeout;
} CUDBGEventCallbackData;

```

```

#pragma pack(push,1)
typedef struct {
    uint32_t dev;
    uint64_t gridId64;
    uint32_t tid;
    uint64_t context;
    uint64_t module;
    uint64_t function;
    uint64_t functionEntry;
    CuDim3   gridDim;
    CuDim3   blockDim;

```

```

        CUDBGKernelType type;
        uint64_t parentGridId;
        CUDBGKernelOrigin origin;
    } CUDBGGridInfo;
#pragma pack(pop)

typedef void (*CUDBGNotifyNewEventCallback31)(void *data);
typedef void (*CUDBGNotifyNewEventCallback40)(CUDBGEventCallbackData40
*data);
typedef void (*CUDBGNotifyNewEventCallback)(CUDBGEventCallbackData *data);

/*----- Exceptions
-----*/

typedef enum {
    CUDBG_EXCEPTION_UNKNOWN = 0xFFFFFFFFFU, // Force
sizeof(CUDBGException_t)==4
    CUDBG_EXCEPTION_NONE = 0,
    CUDBG_EXCEPTION_LANE_ILLEGAL_ADDRESS = 1,
    CUDBG_EXCEPTION_LANE_USER_STACK_OVERFLOW = 2,
    CUDBG_EXCEPTION_DEVICE_HARDWARE_STACK_OVERFLOW = 3,
    CUDBG_EXCEPTION_WARP_ILLEGAL_INSTRUCTION = 4,
    CUDBG_EXCEPTION_WARP_OUT_OF_RANGE_ADDRESS = 5,
    CUDBG_EXCEPTION_WARP_MISALIGNED_ADDRESS = 6,
    CUDBG_EXCEPTION_WARP_INVALID_ADDRESS_SPACE = 7,
    CUDBG_EXCEPTION_WARP_INVALID_PC = 8,
    CUDBG_EXCEPTION_WARP_HARDWARE_STACK_OVERFLOW = 9,
    CUDBG_EXCEPTION_DEVICE_ILLEGAL_ADDRESS = 10,
    CUDBG_EXCEPTION_LANE_MISALIGNED_ADDRESS = 11,
    CUDBG_EXCEPTION_WARP_ASSERT = 12,
    CUDBG_EXCEPTION_LANE_SYSCALL_ERROR = 13,
    CUDBG_EXCEPTION_WARP_ILLEGAL_ADDRESS = 14,
} CUDBGException_t;

/*----- Warp State
-----*/
#pragma pack(push,1)
typedef struct {
    uint64_t virtualPC;
    CuDim3 threadIdx;
    CUDBGException_t exception;
} CUDBGLaneState;

typedef struct {
    uint64_t gridId;
    uint64_t errorPC;
    CuDim3 blockIdx;
    uint32_t validLanes;
    uint32_t activeLanes;
    uint32_t errorPCValid;
    CUDBGLaneState lane[32];
} CUDBGWarpState;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct {
    uint64_t startAddress;
    uint64_t size;
} CUDBGMemoryInfo;
#pragma pack(pop)

/*----- Exports
-----*/

typedef const struct CUDBGAPI_st *CUDBGAPI;

```

```

    CUDBGResult cudbgGetAPI(uint32_t major, uint32_t minor, uint32_t rev,
    CUDBGAPI *api);
    CUDBGResult cudbgGetAPIVersion(uint32_t *major, uint32_t *minor, uint32_t
    *rev);
    CUDBGResult cudbgMain(int apiClientPid, uint32_t apiClientRevision, int
    sessionId, int attachState,
                           int attachEventInitialized, int writeFd, int
    detachFd, int attachStubInUse,
                           int enablePreemptionDebugging);
    void cudbgApiInit(uint32_t arg);
    void cudbgApiAttach(void);
    void cudbgApiDetach(void);
    void CUDBG_REPORT_DRIVER_API_ERROR(void);
    void CUDBG_REPORT_DRIVER_INTERNAL_ERROR(void);

    extern uint32_t CUDBG_IPC_FLAG_NAME;
    extern uint32_t CUDBG_RPC_ENABLED;
    extern uint32_t CUDBG_APICLIENT_PID;
    extern uint32_t CUDBG_I_AM_DEBUGGER;
    extern uint32_t CUDBG_DEBUGGER_INITIALIZED;
    extern uint32_t CUDBG_APICLIENT_REVISION;
    extern uint32_t CUDBG_SESSION_ID;
    extern uint64_t CUDBG_REPORTED_DRIVER_API_ERROR_CODE;
    extern uint64_t CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_SIZE;
    extern uint64_t CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_ADDR;
    extern uint64_t CUDBG_REPORTED_DRIVER_INTERNAL_ERROR_CODE;
    extern uint32_t CUDBG_ATTACH_HANDLER_AVAILABLE;
    extern uint32_t CUDBG_DETACH_SUSPENDED_DEVICES_MASK;
    extern uint32_t CUDBG_ENABLE_LAUNCH_BLOCKING;
    extern uint32_t CUDBG_ENABLE_INTEGRATED_MEMCHECK;
    extern uint32_t CUDBG_ENABLE_PREEMPTION_DEBUGGING;
    extern uint32_t CUDBG_RESUME_FOR_ATTACH_DETACH;
    extern uint32_t CUDBG_REPORT_DRIVER_API_ERROR_FLAGS;

    struct CUDBGAPI_st {
        /* Initialization */
        CUDBGResult (*initialize)(void);
        CUDBGResult (*finalize)(void);

        /* Device Execution Control */
        CUDBGResult (*suspendDevice)(uint32_t dev);
        CUDBGResult (*resumeDevice)(uint32_t dev);
        CUDBGResult (*singleStepWarp40)(uint32_t dev, uint32_t sm, uint32_t
wp);

        /* Breakpoints */
        CUDBGResult (*setBreakpoint31)(uint64_t addr);
        CUDBGResult (*unsetBreakpoint31)(uint64_t addr);

        /* Device State Inspection */
        CUDBGResult (*readGridId50)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t *gridId);
        CUDBGResult (*readBlockIdx32)(uint32_t dev, uint32_t sm, uint32_t wp,
CuDim2 *blockIdx);
        CUDBGResult (*readThreadIdx)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, CuDim3 *threadIdx);
        CUDBGResult (*readBrokenWarps)(uint32_t dev, uint32_t sm, uint64_t
*brokenWarpsMask);
        CUDBGResult (*readValidWarps)(uint32_t dev, uint32_t sm, uint64_t
*validWarpsMask);
        CUDBGResult (*readValidLanes)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t *validLanesMask);
        CUDBGResult (*readActiveLanes)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t *activeLanesMask);
        CUDBGResult (*readCodeMemory)(uint32_t dev, uint64_t addr, void *buf,
uint32_t sz);

```



```

    CUDBGResult (*readConstMemory)(uint32_t dev, uint64_t addr, void *buf,
uint32_t sz);
    CUDBGResult (*readGlobalMemory31)(uint32_t dev, uint64_t addr, void
*buf, uint32_t sz);
    CUDBGResult (*readParamMemory)(uint32_t dev, uint32_t sm, uint32_t wp,
uint64_t addr, void *buf, uint32_t sz);
    CUDBGResult (*readSharedMemory)(uint32_t dev, uint32_t sm, uint32_t
wp, uint64_t addr, void *buf, uint32_t sz);
    CUDBGResult (*readLocalMemory)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint64_t addr, void *buf, uint32_t sz);
    CUDBGResult (*readRegister)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint32_t regno, uint32_t *val);
    CUDBGResult (*readPC)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t
ln, uint64_t *pc);
    CUDBGResult (*readVirtualPC)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint64_t *pc);
    CUDBGResult (*readLaneStatus)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, bool *error);

    /* Device State Alteration */
    CUDBGResult (*writeGlobalMemory31)(uint32_t dev, uint64_t addr, const
void *buf, uint32_t sz);
    CUDBGResult (*writeParamMemory)(uint32_t dev, uint32_t sm, uint32_t
wp, uint64_t addr, const void *buf, uint32_t sz);
    CUDBGResult (*writeSharedMemory)(uint32_t dev, uint32_t sm, uint32_t
wp, uint64_t addr, const void *buf, uint32_t sz);
    CUDBGResult (*writeLocalMemory)(uint32_t dev, uint32_t sm, uint32_t
wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz);
    CUDBGResult (*writeRegister)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint32_t regno, uint32_t val);

    /* Grid Properties */
    CUDBGResult (*getGridDim32)(uint32_t dev, uint32_t sm, uint32_t wp,
CuDim2 *gridDim);
    CUDBGResult (*getBlockDim)(uint32_t dev, uint32_t sm, uint32_t wp,
CuDim3 *blockDim);
    CUDBGResult (*getTID)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t
*tid);
    CUDBGResult (*getElfImage32)(uint32_t dev, uint32_t sm, uint32_t wp,
bool relocated, void **elfImage, uint32_t *size);

    /* Device Properties */
    CUDBGResult (*getDeviceType)(uint32_t dev, char *buf, uint32_t sz);
    CUDBGResult (*getSmType)(uint32_t dev, char *buf, uint32_t sz);
    CUDBGResult (*getNumDevices)(uint32_t *numDev);
    CUDBGResult (*getNumSMs)(uint32_t dev, uint32_t *numSMs);
    CUDBGResult (*getNumWarps)(uint32_t dev, uint32_t *numWarps);
    CUDBGResult (*getNumLanes)(uint32_t dev, uint32_t *numLanes);
    CUDBGResult (*getNumRegisters)(uint32_t dev, uint32_t *numRegs);

    /* DWARF-related routines */
    CUDBGResult (*getPhysicalRegister30)(uint64_t pc, char *reg, uint32_t
*buf, uint32_t sz, uint32_t *numPhysRegs, CUDBGRegClass *regClass);
    CUDBGResult (*disassemble)(uint32_t dev, uint64_t addr, uint32_t
*instSize, char *buf, uint32_t sz);
    CUDBGResult (*isDeviceCodeAddress55)(uintptr_t addr, bool
*isDeviceAddress);
    CUDBGResult (*lookupDeviceCodeSymbol)(char *symName, bool *symFound,
uintptr_t *symAddr);

    /* Events */
    CUDBGResult (*setNotifyNewEventCallback31)
(CUDBGNotifyNewEventCallback31 callback, void *data);
    CUDBGResult (*getNextEvent30)(CUDBGEvent30 *event);
    CUDBGResult (*acknowledgeEvent30)(CUDBGEvent30 *event);

    /* 3.1 Extensions */

```

```

        CUDBGResult (*getGridAttribute)(uint32_t dev, uint32_t sm, uint32_t
wp, CUDBGAttribute attr, uint64_t *value);
        CUDBGResult (*getGridAttributes)(uint32_t dev, uint32_t sm, uint32_t
wp, CUDBGAttributeValuePair *pairs, uint32_t numPairs);
        CUDBGResult (*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);
        CUDBGResult (*readLaneException)(uint32_t dev, uint32_t sm, uint32_t
wp, uint32_t ln, CUDBGException_t *exception);
        CUDBGResult (*getNextEvent32)(CUDBGEvent32 *event);
        CUDBGResult (*acknowledgeEvents42)(void);

/* 3.1 - ABI */
CUDBGResult (*readCallDepth32)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t *depth);
CUDBGResult (*readReturnAddress32)(uint32_t dev, uint32_t sm, uint32_t
wp, uint32_t level, uint64_t *ra);
CUDBGResult (*readVirtualReturnAddress32)(uint32_t dev, uint32_t sm,
uint32_t wp, uint32_t level, uint64_t *ra);

/* 3.2 Extensions */
CUDBGResult (*readGlobalMemory55)(uint32_t dev, uint32_t sm, uint32_t
wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz);
CUDBGResult (*writeGlobalMemory55)(uint32_t dev, uint32_t sm, uint32_t
wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz);
CUDBGResult (*readPinnedMemory)(uint64_t addr, void *buf, uint32_t
sz);
CUDBGResult (*writePinnedMemory)(uint64_t addr, const void *buf,
uint32_t sz);
CUDBGResult (*setBreakpoint)(uint32_t dev, uint64_t addr);
CUDBGResult (*unsetBreakpoint)(uint32_t dev, uint64_t addr);
CUDBGResult (*setNotifyNewEventCallback40)
(CUDBGNotifyNewEventCallback40 callback);

/* 4.0 Extensions */
CUDBGResult (*getNextEvent42)(CUDBGEvent42 *event);
CUDBGResult (*readTextureMemory)(uint32_t devId, uint32_t vsm,
uint32_t wp, uint32_t id, uint32_t dim, uint32_t *coords, void *buf, uint32_t
sz);
CUDBGResult (*readBlockIdx)(uint32_t dev, uint32_t sm, uint32_t wp,
CuDim3 *blockIdx);
CUDBGResult (*getGridDim)(uint32_t dev, uint32_t sm, uint32_t wp,
CuDim3 *gridDim);
CUDBGResult (*readCallDepth)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint32_t *depth);
CUDBGResult (*readReturnAddress)(uint32_t dev, uint32_t sm, uint32_t
wp, uint32_t ln, uint32_t level, uint64_t *ra);
CUDBGResult (*readVirtualReturnAddress)(uint32_t dev, uint32_t sm,
uint32_t wp, uint32_t ln, uint32_t level, uint64_t *ra);
CUDBGResult (*getElfImage)(uint32_t dev, uint32_t sm, uint32_t wp,
bool relocated, void **elfImage, uint64_t *size);

/* 4.1 Extensions */
CUDBGResult (*getHostAddrFromDeviceAddr)(uint32_t dev, uint64_t
device_addr, uint64_t *host_addr);
CUDBGResult (*singleStepWarp)(uint32_t dev, uint32_t sm, uint32_t wp,
uint64_t *warpMask);
CUDBGResult (*setNotifyNewEventCallback)
(CUDBGNotifyNewEventCallback callback);
CUDBGResult (*readSyscallCallDepth)(uint32_t dev, uint32_t sm,
uint32_t wp, uint32_t ln, uint32_t *depth);

/* 4.2 Extensions */
CUDBGResult (*readTextureMemoryBindless)(uint32_t devId, uint32_t vsm,
uint32_t wp, uint32_t texSymtabIndex, uint32_t dim, uint32_t *coords, void
*buf, uint32_t sz);

```

```

/* 5.0 Extensions */
CUDBGResult (*clearAttachState) (void);
CUDBGResult (*getNextSyncEvent50) (CUDBGEvent50 *event);
CUDBGResult (*memcheckReadErrorAddress) (uint32_t dev, uint32_t sm,
uint32_t wp, uint32_t ln, uint64_t *address, ptxStorageKind *storage);
CUDBGResult (*acknowledgeSyncEvents) (void);
CUDBGResult (*getNextAsyncEvent50) (CUDBGEvent50 *event);
CUDBGResult (*requestCleanupOnDetach55) (void);
CUDBGResult (*initializeAttachStub) (void);
CUDBGResult (*getGridStatus50) (uint32_t dev, uint32_t
gridId, CUDBGGridStatus *status);

/* 5.5 Extensions */
CUDBGResult (*getNextSyncEvent55) (CUDBGEvent55 *event);
CUDBGResult (*getNextAsyncEvent55) (CUDBGEvent55 *event);
CUDBGResult (*getGridInfo) (uint32_t dev, uint64_t
gridId64, CUDBGGridInfo *gridInfo);
CUDBGResult (*readGridId) (uint32_t dev, uint32_t sm, uint32_t wp,
uint64_t *gridId64);
CUDBGResult (*getGridStatus) (uint32_t dev, uint64_t
gridId64, CUDBGGridStatus *status);
CUDBGResult (*setKernelLaunchNotificationMode)
(CUDBGKernelLaunchNotifyMode mode);
CUDBGResult (*getDevicePCIBusInfo) (uint32_t devId, uint32_t
*pciBusId, uint32_t *pciDevId);
CUDBGResult (*readDeviceExceptionState) (uint32_t devId, uint64_t
*exceptionSMMask);

/* 6.0 Extensions */
CUDBGResult (*getAdjustedCodeAddress) (uint32_t devId, uint64_t
address, uint64_t *adjustedAddress, CUDBGAdjAddrAction adjAction);
CUDBGResult (*readErrorPC) (uint32_t devId, uint32_t sm, uint32_t wp,
uint64_t *errorPC, bool *errorPCValid);
CUDBGResult (*getNextEvent) (CUDBGEventQueueType type, CUDBGEvent
*event);
CUDBGResult (*getElfImageByHandle) (uint32_t devId, uint64_t handle,
CUDBGElfImageType type, void *elfImage, uint64_t size);
CUDBGResult (*resumeWarpsUntilPC) (uint32_t devId, uint32_t sm,
uint64_t warpMask, uint64_t virtPC);
CUDBGResult (*readWarpState) (uint32_t devId, uint32_t sm, uint32_t wp,
CUDBGWarpState *state);
CUDBGResult (*readRegisterRange) (uint32_t devId, uint32_t sm, uint32_t
wp, uint32_t ln, uint32_t index, uint32_t registers_size, uint32_t *registers);
CUDBGResult (*readGenericMemory) (uint32_t dev, uint32_t sm, uint32_t
wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz);
CUDBGResult (*writeGenericMemory) (uint32_t dev, uint32_t sm, uint32_t
wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz);
CUDBGResult (*readGlobalMemory) (uint64_t addr, void *buf, uint32_t
sz);
CUDBGResult (*writeGlobalMemory) (uint64_t addr, const void *buf,
uint32_t sz);
CUDBGResult (*getManagedMemoryRegionInfo) (uint64_t startAddress,
CUDBGMemoryInfo *memoryInfo, uint32_t memoryInfo_size, uint32_t *numEntries);
CUDBGResult (*isDeviceCodeAddress) (uintptr_t addr, bool
*isDeviceAddress);
CUDBGResult (*requestCleanupOnDetach) (uint32_t appResumeFlag);

/* 6.5 Extensions */
CUDBGResult (*readPredicates) (uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint32_t predicates_size, uint32_t *predicates);
CUDBGResult (*writePredicates) (uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint32_t predicates_size, const uint32_t *predicates);
CUDBGResult (*getNumPredicates) (uint32_t dev, uint32_t
*numPredicates);
CUDBGResult (*readCCRegister) (uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint32_t *val);

```

```

        CUDBGResult (*writeCCRegister)(uint32_t dev, uint32_t sm, uint32_t wp,
uint32_t ln, uint32_t val);

        CUDBGResult (*getDeviceName)(uint32_t dev, char *buf, uint32_t sz);
    };

#ifdef __cplusplus
}
#endif

#endif

```

## struct CUDBGAPI\_st

The CUDA debugger API routines.

## 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.

## struct CUDBGGridInfo

Grid info.

# Chapter 7.

## GLOBALS

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

**CUDBG\_ADJ\_CURRENT\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_ADJ\_NEXT\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_ADJ\_PREVIOUS\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_ATTR\_GRID\_LAUNCH\_BLOCKING**  
[cudadebugger.h](#)

**CUDBG\_ATTR\_GRID\_TID**  
[cudadebugger.h](#)

**CUDBG\_ELF\_IMAGE\_PROPERTIES\_SYSTEM**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_ALL\_DEVICES\_WATCHDOGGED**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_AMBIGUOUS\_MEMORY\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_ATTACH\_NOT\_POSSIBLE**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_BUFFER\_TOO\_SMALL**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_COMMUNICATION\_FAILURE**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_FORK\_FAILED**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_INCOMPATIBLE\_API**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_INCOMPATIBLE\_DISPLAY\_DRIVER**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INITIALIZATION\_FAILURE**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INTERNAL**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_ADDRESS**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_ARGS**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_ATTRIBUTE**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_CALL\_LEVEL**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_CONTEXT**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_COORDINATES**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_DEVICE**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_ENVVAR\_ARGS**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_GRID**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_LANE**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_MEMORY\_SEGMENT**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_MODULE**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_SM**

[cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_WARP**

[cudadebugger.h](#)

**CUDBG\_ERROR\_LANE\_NOT\_IN\_SYSCALL**

[cudadebugger.h](#)

**CUDBG\_ERROR\_MEMCHECK\_NOT\_ENABLED**

[cudadebugger.h](#)

**CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED**

[cudadebugger.h](#)

**CUDBG\_ERROR\_MEMORY\_UNMAPPING\_FAILED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_NO\_DEVICE\_AVAILABLE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_OS\_RESOURCES**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_RUNNING\_DEVICE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_SOME\_DEVICES\_WATCHDOGGED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_SUSPENDED\_DEVICE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_UNINITIALIZED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_UNKNOWN**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_UNKNOWN\_FUNCTION**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_ZERO\_CALL\_DEPTH**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_ATTACH\_COMPLETE**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_CTX\_CREATE**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_CTX\_DESTROY**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_CTX\_POP**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_CTX\_PUSH**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_ELF\_IMAGE\_LOADED**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_INTERNAL\_ERROR**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_INVALID**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_KERNEL\_FINISHED**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_KERNEL\_READY**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_TIMEOUT**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_DEVICE\_HARDWARE\_STACK\_OVERFLOW**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_DEVICE\_ILLEGAL\_ADDRESS**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_LANE\_ILLEGAL\_ADDRESS**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_LANE\_MISALIGNED\_ADDRESS**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_LANE\_USER\_STACK\_OVERFLOW**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_NONE**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_UNKNOWN**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_HARDWARE\_STACK\_OVERFLOW**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_ILLEGAL\_INSTRUCTION**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_INVALID\_ADDRESS\_SPACE**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_INVALID\_PC**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_MISALIGNED\_ADDRESS**  
    [cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_OUT\_OF\_RANGE\_ADDRESS**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_ACTIVE**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_INVALID**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_PENDING**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_SLEEPING**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_TERMINATED**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_UNDETERMINED**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_LAUNCH\_NOTIFY\_EVENT**  
    [cudadebugger.h](#)



**CUDBG\_KNL\_ORIGIN\_CPU**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_ORIGIN\_GPU**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_TYPE\_APPLICATION**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_TYPE\_SYSTEM**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_TYPE\_UNKNOWN**  
    [cudadebugger.h](#)

**CUDBG\_SUCCESS**  
    [cudadebugger.h](#)

**CUDBGAdjAddrAction**  
    [cudadebugger.h](#)

**CUDBGAttribute**  
    [cudadebugger.h](#)

**CUDBGElfImageProperties**  
    [cudadebugger.h](#)

**CUDBGEventKind**  
    [cudadebugger.h](#)

**CUDBGException\_t**  
    [cudadebugger.h](#)

**cudbgGetAPIVersion()**  
    [cudadebugger.h](#)

**CUDBGGridStatus**  
    [cudadebugger.h](#)

**CUDBGKernelLaunchNotifyMode**  
    [cudadebugger.h](#)

**CUDBGKernelOrigin**  
    [cudadebugger.h](#)

**CUDBGKernelType**  
    [cudadebugger.h](#)

**CUDBGNotifyNewEventCallback**  
    [cudadebugger.h](#)

**CUDBGNotifyNewEventCallback31**  
    [cudadebugger.h](#)

**CUDBGRegClass**  
    [cudadebugger.h](#)

**CUDBGResult**  
    [cudadebugger.h](#)

**REG\_CLASS\_INVALID**  
    [cudadebugger.h](#)

**REG\_CLASS\_LMEM\_REG\_OFFSET**

[cudadebugger.h](#)

**REG\_CLASS\_MEM\_LOCAL**

[cudadebugger.h](#)

**REG\_CLASS\_REG\_ADDR**

[cudadebugger.h](#)

**REG\_CLASS\_REG\_CC**

[cudadebugger.h](#)

**REG\_CLASS\_REG\_FULL**

[cudadebugger.h](#)

**REG\_CLASS\_REG\_HALF**

[cudadebugger.h](#)

**REG\_CLASS\_REG\_PRED**

[cudadebugger.h](#)

## 7.1. Globals - Functions

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

**cudbgGetAPIVersion()**

[cudadebugger.h](#)

## 7.2. Globals - Typedefs

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

**CUDBGNotifyNewEventCallback**

[cudadebugger.h](#)

**CUDBGNotifyNewEventCallback31**

[cudadebugger.h](#)

## 7.3. Globals - Enumerations

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

**CUDBGAdjAddrAction**

[cudadebugger.h](#)

**CUDBGAttribute**

[cudadebugger.h](#)

**CUDBGElfImageProperties**  
[cudadebugger.h](#)  
**CUDBGEventKind**  
[cudadebugger.h](#)  
**CUDBGException\_t**  
[cudadebugger.h](#)  
**CUDBGGridStatus**  
[cudadebugger.h](#)  
**CUDBGKernelLaunchNotifyMode**  
[cudadebugger.h](#)  
**CUDBGKernelOrigin**  
[cudadebugger.h](#)  
**CUDBGKernelType**  
[cudadebugger.h](#)  
**CUDBGRegClass**  
[cudadebugger.h](#)  
**CUDBGResult**  
[cudadebugger.h](#)

## 7.4. Globals - Enumerator

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

**CUDBG\_ADJ\_CURRENT\_ADDRESS**  
[cudadebugger.h](#)  
**CUDBG\_ADJ\_NEXT\_ADDRESS**  
[cudadebugger.h](#)  
**CUDBG\_ADJ\_PREVIOUS\_ADDRESS**  
[cudadebugger.h](#)  
**CUDBG\_ATTR\_GRID\_LAUNCH\_BLOCKING**  
[cudadebugger.h](#)  
**CUDBG\_ATTR\_GRID\_TID**  
[cudadebugger.h](#)  
**CUDBG\_ELF\_IMAGE\_PROPERTIES\_SYSTEM**  
[cudadebugger.h](#)  
**CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM**  
[cudadebugger.h](#)  
**CUDBG\_ERROR\_ALL\_DEVICES\_WATCHDOGGED**  
[cudadebugger.h](#)  
**CUDBG\_ERROR\_AMBIGUOUS\_MEMORY\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_ERROR\_ATTACH\_NOT\_POSSIBLE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_BUFFER\_TOO\_SMALL**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_COMMUNICATION\_FAILURE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_FORK\_FAILED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INCOMPATIBLE\_API**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INCOMPATIBLE\_DISPLAY\_DRIVER**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INITIALIZATION\_FAILURE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INTERNAL**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_ADDRESS**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_ARGS**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_ATTRIBUTE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_CALL\_LEVEL**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_CONTEXT**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_COORDINATES**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_DEVICE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_ENVVAR\_ARGS**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_GRID**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_LANE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_MEMORY\_SEGMENT**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_MODULE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_SM**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_INVALID\_WARP**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_LANE\_NOT\_IN\_SYSCALL**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_MEMCHECK\_NOT\_ENABLED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_MEMORY\_UNMAPPING\_FAILED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_NO\_DEVICE\_AVAILABLE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_OS\_RESOURCES**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_RUNNING\_DEVICE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_SOME\_DEVICES\_WATCHDOGGED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_SUSPENDED\_DEVICE**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_UNINITIALIZED**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_UNKNOWN**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_UNKNOWN\_FUNCTION**  
    [cudadebugger.h](#)

**CUDBG\_ERROR\_ZERO\_CALL\_DEPTH**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_ATTACH\_COMPLETE**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_CTX\_CREATE**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_CTX\_DESTROY**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_CTX\_POP**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_CTX\_PUSH**  
    [cudadebugger.h](#)

**CUDBG\_EVENT\_ELF\_IMAGE\_LOADED**  
[cudadebugger.h](#)

**CUDBG\_EVENT\_INTERNAL\_ERROR**  
[cudadebugger.h](#)

**CUDBG\_EVENT\_INVALID**  
[cudadebugger.h](#)

**CUDBG\_EVENT\_KERNEL\_FINISHED**  
[cudadebugger.h](#)

**CUDBG\_EVENT\_KERNEL\_READY**  
[cudadebugger.h](#)

**CUDBG\_EVENT\_TIMEOUT**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_DEVICE\_HARDWARE\_STACK\_OVERFLOW**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_DEVICE\_ILLEGAL\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_LANE\_ILLEGAL\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_LANE\_MISALIGNED\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_LANE\_USER\_STACK\_OVERFLOW**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_NONE**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_UNKNOWN**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_HARDWARE\_STACK\_OVERFLOW**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_ILLEGAL\_INSTRUCTION**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_INVALID\_ADDRESS\_SPACE**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_INVALID\_PC**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_MISALIGNED\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_EXCEPTION\_WARP\_OUT\_OF\_RANGE\_ADDRESS**  
[cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_ACTIVE**  
[cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_INVALID**  
[cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_PENDING**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_SLEEPING**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_TERMINATED**  
    [cudadebugger.h](#)

**CUDBG\_GRID\_STATUS\_UNDETERMINED**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_LAUNCH\_NOTIFY\_EVENT**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_ORIGIN\_CPU**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_ORIGIN\_GPU**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_TYPE\_APPLICATION**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_TYPE\_SYSTEM**  
    [cudadebugger.h](#)

**CUDBG\_KNL\_TYPE\_UNKNOWN**  
    [cudadebugger.h](#)

**CUDBG\_SUCCESS**  
    [cudadebugger.h](#)

**REG\_CLASS\_INVALID**  
    [cudadebugger.h](#)

**REG\_CLASS\_LMEM\_REG\_OFFSET**  
    [cudadebugger.h](#)

**REG\_CLASS\_MEM\_LOCAL**  
    [cudadebugger.h](#)

**REG\_CLASS\_REG\_ADDR**  
    [cudadebugger.h](#)

**REG\_CLASS\_REG\_CC**  
    [cudadebugger.h](#)

**REG\_CLASS\_REG\_FULL**  
    [cudadebugger.h](#)

**REG\_CLASS\_REG\_HALF**  
    [cudadebugger.h](#)

**REG\_CLASS\_REG\_PRED**  
    [cudadebugger.h](#)

## Chapter 8.

# 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::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**

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

## **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**

© 2007-2014 NVIDIA Corporation. All rights reserved.