



# NVIDIA CUDA DEBUGGER API

Reference Manual

Version 5.0

---

October 2012





## **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 or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

## **Trademarks**

NVIDIA, the NVIDIA logo, GeForce, Tesla, and Quadro are trademarks or registered trademarks of NVIDIA Corporation. Other company and product names may be trademarks of the respective companies with which they are associated.

## **Copyright**

© 2007-2012 NVIDIA Corporation. All rights reserved.



**NVIDIA**

NVIDIA Corporation  
2701 San Tomas Expressway  
Santa Clara, CA 95050  
[www.nvidia.com](http://www.nvidia.com)



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Debugger API	1
1.2	ELF and DWARF	2
1.3	ABI Support	3
1.4	Exception Reporting	3
<b>2</b>	<b>Module Index</b>	<b>5</b>
2.1	Modules	5
<b>3</b>	<b>Data Structure Index</b>	<b>7</b>
3.1	Data Structures	7
<b>4</b>	<b>File Index</b>	<b>9</b>
4.1	File List	9
<b>5</b>	<b>Module Documentation</b>	<b>11</b>
5.1	Initialization	11
5.1.1	Detailed Description	11
5.1.2	Variable Documentation	11
5.1.2.1	finalize	11
5.1.2.2	initialize	11
5.2	Device Execution Control	13
5.2.1	Detailed Description	13
5.2.2	Variable Documentation	13
5.2.2.1	resumeDevice	13
5.2.2.2	singleStepWarp	13
5.2.2.3	singleStepWarp40	14
5.2.2.4	suspendDevice	14
5.3	Breakpoints	16
5.3.1	Detailed Description	16

5.3.2	Variable Documentation	16
5.3.2.1	setBreakpoint	16
5.3.2.2	setBreakpoint31	16
5.3.2.3	unsetBreakpoint	17
5.3.2.4	unsetBreakpoint31	17
5.4	Device State Inspection	18
5.4.1	Detailed Description	20
5.4.2	Variable Documentation	20
5.4.2.1	memcheckReadErrorAddress	20
5.4.2.2	readActiveLanes	21
5.4.2.3	readBlockIdx	21
5.4.2.4	readBlockIdx32	22
5.4.2.5	readBrokenWarps	22
5.4.2.6	readCallDepth	23
5.4.2.7	readCallDepth32	23
5.4.2.8	readCodeMemory	24
5.4.2.9	readConstMemory	24
5.4.2.10	readGlobalMemory	25
5.4.2.11	readGlobalMemory31	26
5.4.2.12	readGridId	26
5.4.2.13	readLaneException	27
5.4.2.14	readLaneStatus	27
5.4.2.15	readLocalMemory	28
5.4.2.16	readParamMemory	28
5.4.2.17	readPC	29
5.4.2.18	readPinnedMemory	30
5.4.2.19	readRegister	30
5.4.2.20	readReturnAddress	31
5.4.2.21	readReturnAddress32	31
5.4.2.22	readSharedMemory	32
5.4.2.23	readSyscallCallDepth	33
5.4.2.24	readTextureMemory	33
5.4.2.25	readTextureMemoryBindless	34
5.4.2.26	readThreadId	35
5.4.2.27	readValidLanes	35
5.4.2.28	readValidWarps	36
5.4.2.29	readVirtualPC	36

---

5.4.2.30	readVirtualReturnAddress	37
5.4.2.31	readVirtualReturnAddress32	37
5.4.2.32	writePinnedMemory	38
5.5	Device State Alteration	39
5.5.1	Detailed Description	39
5.5.2	Variable Documentation	39
5.5.2.1	writeGlobalMemory	39
5.5.2.2	writeGlobalMemory31	40
5.5.2.3	writeLocalMemory	40
5.5.2.4	writeParamMemory	41
5.5.2.5	writeRegister	42
5.5.2.6	writeSharedMemory	42
5.6	Grid Properties	44
5.6.1	Detailed Description	44
5.6.2	Variable Documentation	44
5.6.2.1	getBlockDim	44
5.6.2.2	getElfImage	45
5.6.2.3	getElfImage32	45
5.6.2.4	getGridAttribute	46
5.6.2.5	getGridAttributes	46
5.6.2.6	getGridDim	46
5.6.2.7	getGridDim32	47
5.6.2.8	getTID	47
5.7	Device Properties	48
5.7.1	Detailed Description	48
5.7.2	Variable Documentation	48
5.7.2.1	getDeviceType	48
5.7.2.2	getNumDevices	49
5.7.2.3	getNumLanes	49
5.7.2.4	getNumRegisters	49
5.7.2.5	getNumSMs	50
5.7.2.6	getNumWarps	50
5.7.2.7	getSmType	51
5.8	DWARF Utilities	52
5.8.1	Detailed Description	52
5.8.2	Variable Documentation	52
5.8.2.1	disassemble	52

5.8.2.2	getHostAddrFromDeviceAddr	53
5.8.2.3	getPhysicalRegister30	53
5.8.2.4	getPhysicalRegister40	53
5.8.2.5	isDeviceCodeAddress	54
5.8.2.6	lookupDeviceCodeSymbol	54
5.9	Events	55
5.9.1	Detailed Description	56
5.9.2	Enumeration Type Documentation	57
5.9.2.1	CUDBGEventKind	57
5.9.3	Variable Documentation	58
5.9.3.1	acknowledgeEvent30	58
5.9.3.2	acknowledgeEvents42	58
5.9.3.3	acknowledgeSyncEvents	58
5.9.3.4	getNextAsyncEvent	58
5.9.3.5	getNextEvent30	59
5.9.3.6	getNextEvent32	59
5.9.3.7	getNextEvent42	59
5.9.3.8	getNextSyncEvent	59
5.9.3.9	setNotifyNewEventCallback	60
5.9.3.10	setNotifyNewEventCallback31	60
5.9.3.11	setNotifyNewEventCallback40	60
<b>6</b>	<b>Data Structure Documentation</b>	<b>61</b>
6.1	cudbgGetAPI Struct Reference	61
6.1.1	Detailed Description	66
6.1.2	Field Documentation	67
6.1.2.1	clearAttachState	67
6.1.2.2	getGridStatus	67
6.1.2.3	requestCleanupOnDetach	67
6.2	CUDBGEvent Struct Reference	68
6.2.1	Detailed Description	68
6.3	CUDBGEvent30 Struct Reference	69
6.3.1	Detailed Description	69
6.4	CUDBGEvent30::CUDBGEvent30::cases30_st Union Reference	70
6.4.1	Detailed Description	70
6.5	CUDBGEvent30::CUDBGEvent30::cases30_st::CUDBGEvent30::cases30_st::elfImageLoaded30_st Struct Reference	71

---

6.5.1	Detailed Description	71
6.6	CUDBGEvent30::CUDBGEvent30::cases30_st::CUDBGEvent30::cases30_st::kernelFinished30_st Struct Reference	72
6.6.1	Detailed Description	72
6.7	CUDBGEvent30::CUDBGEvent30::cases30_st::CUDBGEvent30::cases30_st::kernelReady30_st Struct Reference	73
6.7.1	Detailed Description	73
6.8	CUDBGEvent32 Struct Reference	74
6.8.1	Detailed Description	74
6.9	CUDBGEvent32::CUDBGEvent32::cases32_st Union Reference	75
6.9.1	Detailed Description	75
6.10	CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::contextCreate32_st Struct Reference	76
6.10.1	Detailed Description	76
6.11	CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::contextDestroy32_st Struct Reference	77
6.11.1	Detailed Description	77
6.12	CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::contextPop32_st Struct Reference	78
6.12.1	Detailed Description	78
6.13	CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::contextPush32_st Struct Reference	79
6.13.1	Detailed Description	79
6.14	CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::elfImageLoaded32_st Struct Reference	80
6.14.1	Detailed Description	80
6.15	CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::kernelFinished32_st Struct Reference	81
6.15.1	Detailed Description	81
6.16	CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::kernelReady32_st Struct Reference	82
6.16.1	Detailed Description	82
6.17	CUDBGEvent42 Struct Reference	83
6.17.1	Detailed Description	83
6.18	CUDBGEvent42::CUDBGEvent42::cases42_st Union Reference	84
6.18.1	Detailed Description	84
6.19	CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextCreate42_st Struct Reference	85
6.19.1	Detailed Description	85

6.20	CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextDestroy42_st Struct Reference . . . . .	86
6.20.1	Detailed Description . . . . .	86
6.21	CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextPop42_st Struct Reference . . . . .	87
6.21.1	Detailed Description . . . . .	87
6.22	CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextPush42_st Struct Reference . . . . .	88
6.22.1	Detailed Description . . . . .	88
6.23	CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::elfImageLoaded42_st Struct Reference . . . . .	89
6.23.1	Detailed Description . . . . .	89
6.24	CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::kernelFinished42_st Struct Reference . . . . .	90
6.24.1	Detailed Description . . . . .	90
6.25	CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::kernelReady42_st Struct Reference . . . . .	91
6.25.1	Detailed Description . . . . .	91
6.26	CUDBGEvent::CUDBGEvent::cases_st Union Reference . . . . .	92
6.26.1	Detailed Description . . . . .	92
6.27	CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextCreate_st Struct Reference . . . . .	93
6.27.1	Detailed Description . . . . .	93
6.28	CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextDestroy_st Struct Reference . . . . .	94
6.28.1	Detailed Description . . . . .	94
6.29	CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPop_st Struct Reference . . . . .	95
6.29.1	Detailed Description . . . . .	95
6.30	CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPush_st Struct Reference . . . . .	96
6.30.1	Detailed Description . . . . .	96
6.31	CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st Struct Reference . . . . .	97
6.31.1	Detailed Description . . . . .	97
6.32	CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::internalError_st Struct Reference . . . . .	98
6.32.1	Detailed Description . . . . .	98
6.33	CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelFinished_st Struct Reference . . . . .	99
6.33.1	Detailed Description . . . . .	99
6.34	CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st Struct Reference . . . . .	100
6.34.1	Detailed Description . . . . .	100
6.35	CUDBGEventCallbackData Struct Reference . . . . .	101
6.35.1	Detailed Description . . . . .	101
6.36	CUDBGEventCallbackData40 Struct Reference . . . . .	102

---

6.36.1 Detailed Description . . . . .	102
<b>7 File Documentation</b>	<b>103</b>
7.1 cudadebugger.h File Reference . . . . .	103
7.1.1 Detailed Description . . . . .	107
7.1.2 Enumeration Type Documentation . . . . .	107
7.1.2.1 CUDBGAttribute . . . . .	107
7.1.2.2 CUDBGException_t . . . . .	107
7.1.2.3 CUDBGGridStatus . . . . .	108
7.1.2.4 CUDBGKernelType . . . . .	108
7.1.2.5 CUDBGRegClass . . . . .	108
7.1.2.6 CUDBGResult . . . . .	109
7.1.3 Function Documentation . . . . .	110
7.1.3.1 cudbgGetAPIVersion . . . . .	110



# Chapter 1

## 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.

### 1.1 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.

## 1.2 ELF and DWARF

CUDA applications are compiled in ELF binary format.

DWARF device information is obtained through a [CUDBGEvent](#) of type `CUDBG_EVENT_ELF_IMAGE_LOADED`. This means that the information is not available until runtime, after the CUDA driver has loaded.

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      (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\(\)](#)

## 1.3 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".

## 1.4 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).



# Chapter 2

## Module Index

### 2.1 Modules

Here is a list of all modules:

Initialization . . . . .	11
Device Execution Control . . . . .	13
Breakpoints . . . . .	16
Device State Inspection . . . . .	18
Device State Alteration . . . . .	39
Grid Properties . . . . .	44
Device Properties . . . . .	48
DWARF Utilities . . . . .	52
Events . . . . .	55



# Chapter 3

## Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">cudbgGetAPI</a> (The CUDA debugger API routines ) . . . . .	61
<a href="#">CUDBGEvent</a> (Event information container ) . . . . .	68
<a href="#">CUDBGEvent30</a> (Event information container. Deprecated in 3.1 ) . . . . .	69
<a href="#">CUDBGEvent30::CUDBGEvent30::cases30_st</a> . . . . .	70
<a href="#">CUDBGEvent30::CUDBGEvent30::cases30_st::CUDBGEvent30::cases30_st::elfImageLoaded30_st</a> . . . . .	71
<a href="#">CUDBGEvent30::CUDBGEvent30::cases30_st::CUDBGEvent30::cases30_st::kernelFinished30_st</a> . . . . .	72
<a href="#">CUDBGEvent30::CUDBGEvent30::cases30_st::CUDBGEvent30::cases30_st::kernelReady30_st</a> . . . . .	73
<a href="#">CUDBGEvent32</a> (Event information container. Deprecated in 4.0 ) . . . . .	74
<a href="#">CUDBGEvent32::CUDBGEvent32::cases32_st</a> . . . . .	75
<a href="#">CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::contextCreate32_st</a> . . . . .	76
<a href="#">CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::contextDestroy32_st</a> . . . . .	77
<a href="#">CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::contextPop32_st</a> . . . . .	78
<a href="#">CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::contextPush32_st</a> . . . . .	79
<a href="#">CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::elfImageLoaded32_st</a> . . . . .	80
<a href="#">CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::kernelFinished32_st</a> . . . . .	81
<a href="#">CUDBGEvent32::CUDBGEvent32::cases32_st::CUDBGEvent32::cases32_st::kernelReady32_st</a> . . . . .	82
<a href="#">CUDBGEvent42</a> (Event information container. Deprecated in 4.2 ) . . . . .	83
<a href="#">CUDBGEvent42::CUDBGEvent42::cases42_st</a> . . . . .	84
<a href="#">CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextCreate42_st</a> . . . . .	85
<a href="#">CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextDestroy42_st</a> . . . . .	86
<a href="#">CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextPop42_st</a> . . . . .	87
<a href="#">CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextPush42_st</a> . . . . .	88
<a href="#">CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::elfImageLoaded42_st</a> . . . . .	89
<a href="#">CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::kernelFinished42_st</a> . . . . .	90
<a href="#">CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::kernelReady42_st</a> . . . . .	91
<a href="#">CUDBGEvent::CUDBGEvent::cases_st</a> . . . . .	92
<a href="#">CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextCreate_st</a> . . . . .	93
<a href="#">CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextDestroy_st</a> . . . . .	94
<a href="#">CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPop_st</a> . . . . .	95
<a href="#">CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPush_st</a> . . . . .	96
<a href="#">CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st</a> . . . . .	97
<a href="#">CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::internalError_st</a> . . . . .	98
<a href="#">CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelFinished_st</a> . . . . .	99

---

<a href="#">CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st</a> . . . . .	100
<a href="#">CUDBGEventCallbackData</a> (Event information passed to callback set with setNotifyNewEventCallback function) . . . . .	101
<a href="#">CUDBGEventCallbackData40</a> (Event information passed to callback set with setNotifyNewEventCallback function. Deprecated in 4.1) . . . . .	102

# Chapter 4

## File Index

### 4.1 File List

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

<a href="#">cudadebugger.h</a> (Header file for the CUDA debugger API) . . . . .	103
--	-----



# Chapter 5

## Module Documentation

### 5.1 Initialization

#### Variables

- [CUDBGResult\(\\* CUDBGAPI\\_st::finalize\)](#)(void)  
*Finalize the API and free all memory.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::initialize\)](#)(void)  
*Initialize the API.*

#### 5.1.1 Detailed Description

#### 5.1.2 Variable Documentation

##### 5.1.2.1 [cudbgGetAPI::finalize](#) [inherited]

Finalize the API and free all memory.

#### Returns:

[CUDBG\\_SUCCESS](#),  
[CUDBG\\_ERROR\\_UNINITIALIZED](#),  
[CUDBG\\_ERROR\\_COMMUNICATION\\_FAILURE](#),  
[CUDBG\\_ERROR\\_UNKNOWN](#)

#### See also:

[initialize](#)

##### 5.1.2.2 [cudbgGetAPI::initialize](#) [inherited]

Initialize the API.

**Returns:**

CUDBG\_SUCCESS,  
CUDBG\_ERROR\_UNKNOWN

**See also:**

[finalize](#)

## 5.2 Device Execution Control

### Variables

- [CUDBGResult\(\\* CUDBGAPI\\_st::resumeDevice \)\(uint32\\_t dev\)](#)  
*Resume a suspended CUDA device.*
- [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.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::singleStepWarp40 \)\(uint32\\_t dev, uint32\\_t sm, uint32\\_t wp\)](#)  
*(DEPRECATED)Single step an individual warp on a suspended CUDA device. This function has been deprecated. Use [singleStepWarp\(\)](#) instead.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::suspendDevice \)\(uint32\\_t dev\)](#)  
*Suspends a running CUDA device.*

### 5.2.1 Detailed Description

### 5.2.2 Variable Documentation

#### 5.2.2.1 [cudbgGetAPI::resumeDevice](#) [inherited]

Resume a suspended CUDA device.

#### Parameters:

*dev* - device index

#### Returns:

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

#### See also:

[suspendDevice](#)  
[singleStepWarp](#)

#### 5.2.2.2 [cudbgGetAPI::singleStepWarp](#) [inherited]

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

**See also:**

[resumeDevice](#)  
[suspendDevice](#)

### 5.2.2.3 `cudaGetAPI::singleStepWarp40` [inherited]

(DEPRECATED)Single step an individual warp on a suspended CUDA device. This function has been deprecated. Use [singleStepWarp\(\)](#) instead.

**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

**See also:**

[resumeDevice](#)  
[suspendDevice](#)  
[singleStepWarp](#)

### 5.2.2.4 `cudaGetAPI::suspendDevice` [inherited]

Suspends a running CUDA device.

**Parameters:**

*dev* - device index

**Returns:**

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

**See also:**

[resumeDevice](#)  
[singleStepWarp](#)

## 5.3 Breakpoints

### Variables

- [CUDBGResult\(\\* CUDBGAPI\\_st::setBreakpoint\)\(uint32\\_t dev, uint64\\_t addr\)](#)  
*Sets a breakpoint at the given instruction address for the given device.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::setBreakpoint31\)\(uint64\\_t addr\)](#)  
*Sets a breakpoint at the given instruction address. Deprecated in 3.2.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::unsetBreakpoint\)\(uint32\\_t dev, uint64\\_t addr\)](#)  
*Unsets a breakpoint at the given instruction address for the given device.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::unsetBreakpoint31\)\(uint64\\_t addr\)](#)  
*Unsets a breakpoint at the given instruction address. Deprecated in 3.2.*

### 5.3.1 Detailed Description

### 5.3.2 Variable Documentation

#### 5.3.2.1 `cudbgGetAPI::setBreakpoint` [inherited]

Sets 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

#### See also:

[unsetBreakpoint](#)

#### 5.3.2.2 `cudbgGetAPI::setBreakpoint31` [inherited]

Sets a breakpoint at the given instruction address. Deprecated in 3.2.

#### Parameters:

*addr* - instruction address

#### Returns:

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

See also:

[unsetBreakpoint31](#)

### 5.3.2.3 cudbgGetAPI::unsetBreakpoint [inherited]

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

See also:

[setBreakpoint](#)

### 5.3.2.4 cudbgGetAPI::unsetBreakpoint31 [inherited]

Unsets a breakpoint at the given instruction address. Deprecated in 3.2.

**Parameters:**

*addr* - instruction address

**Returns:**

CUDBG\_SUCCESS,  
CUDBG\_ERROR\_UNINITIALIZED

See also:

[setBreakpoint31](#)

## 5.4 Device State Inspection

### Variables

- 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.*
- 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.*
- 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.*
- 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. Deprecated in 4.0.*
- 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.*
- 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.*
- 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. Deprecated in 4.0.*
- CUDBGResult(\* CUDBGAPI\_st::readCodeMemory )**(uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)  
*Reads content at address in the code memory segment.*
- CUDBGResult(\* CUDBGAPI\_st::readConstMemory )**(uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)  
*Reads content at address in the constant memory segment.*
- CUDBGResult(\* CUDBGAPI\_st::readGlobalMemory )**(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+).*
- CUDBGResult(\* CUDBGAPI\_st::readGlobalMemory31 )**(uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)  
*Reads content at address in the global memory segment. Deprecated in 3.2.*
- CUDBGResult(\* CUDBGAPI\_st::readGridId )**(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*gridId)  
*Reads the CUDA grid index running on a valid warp.*
- 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.*

- `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`.*
- `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.*
- `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.*
- `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.*
- `CUDBGResult(* CUDBGAPI_st::readPinnedMemory )(uint64_t addr, void *buf, uint32_t sz)`  
*Reads content at pinned address in system memory.*
- `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.*
- `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.*
- `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. Deprecated in 4.0.*
- `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.*
- `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.*
- `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.*
- `CUDBGResult(* CUDBGAPI_st::readTextureMemoryBindless )(uint32_t devId, uint32_t vsm, uint32_t wp, uint32_t texSyntabIndex, uint32_t dim, uint32_t *coords, void *buf, uint32_t sz)`  
*Read the content of texture memory with given syntab index and coords on sm\_30 and higher.*
- `CUDBGResult(* CUDBGAPI_st::readThreadId )(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CuDim3 *threadIdx)`  
*Reads the CUDA thread index running on valid lane.*

- `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.*
- `CUDBGResult(* CUDBGAPI_st::readValidWarps )(uint32_t dev, uint32_t sm, uint64_t *validWarpsMask)`  
*Reads the bitmask of valid warps on a given SM.*
- `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.*
- `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.*
- `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. Deprecated in 4.0.*
- `CUDBGResult(* CUDBGAPI_st::writePinnedMemory )(uint64_t addr, const void *buf, uint32_t sz)`  
*Writes content to pinned address in system memory.*

## 5.4.1 Detailed Description

## 5.4.2 Variable Documentation

### 5.4.2.1 `cudbgGetAPI::memcheckReadErrorAddress` [inherited]

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`

### 5.4.2.2 `cudbgGetAPI::readActiveLanes` [inherited]

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

#### See also:

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

### 5.4.2.3 `cudbgGetAPI::readBlockIdx` [inherited]

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

#### See also:

[readGridId](#)  
[readThreadIdx](#)  
[readBrokenWarps](#)

[readValidWarps](#)  
[readValidLanes](#)  
[readActiveLanes](#)

#### 5.4.2.4 `cudaGetAPI::readBlockIdx32` [inherited]

Reads the two-dimensional CUDA block index running on a valid warp. Deprecated in 4.0.

##### 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

##### See also:

[readGridId](#)  
[readThreadId](#)  
[readBrokenWarps](#)  
[readValidWarps](#)  
[readValidLanes](#)  
[readActiveLanes](#)

#### 5.4.2.5 `cudaGetAPI::readBrokenWarps` [inherited]

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

**See also:**

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

**5.4.2.6 cudbgGetAPI::readCallDepth** [inherited]

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

**See also:**

[readReturnAddress](#)  
[readVirtualReturnAddress](#)

**5.4.2.7 cudbgGetAPI::readCallDepth32** [inherited]

Reads the call depth (number of calls) for a given warp. Deprecated in 4.0.

**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

**See also:**

[readReturnAddress32](#)  
[readVirtualReturnAddress32](#)

#### 5.4.2.8 `cudbgGetAPI::readCodeMemory` [inherited]

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

**See also:**

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

#### 5.4.2.9 `cudbgGetAPI::readConstMemory` [inherited]

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

**See also:**

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

**5.4.2.10 cudbgGetAPI::readGlobalMemory** [inherited]

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

**See also:**

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

#### 5.4.2.11 `cuDBGGetAPI::readGlobalMemory31` [inherited]

Reads content at address in the global memory segment. Deprecated in 3.2.

##### 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

##### See also:

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

#### 5.4.2.12 `cuDBGGetAPI::readGridId` [inherited]

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

##### See also:

[readBlockIdx](#)

[readThreadId](#)  
[readBrokenWarps](#)  
[readValidWarps](#)  
[readValidLanes](#)  
[readActiveLanes](#)

#### 5.4.2.13 `cudaGetAPI::readLaneException` [inherited]

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

#### 5.4.2.14 `cudaGetAPI::readLaneStatus` [inherited]

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

#### 5.4.2.15 `cudaGetAPI::readLocalMemory` [inherited]

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

##### See also:

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

#### 5.4.2.16 `cudaGetAPI::readParamMemory` [inherited]

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

**See also:**

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

**5.4.2.17 cudbgGetAPI::readPC** [inherited]

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

**See also:**

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

#### 5.4.2.18 `cuDBGGetAPI::readPinnedMemory` [inherited]

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

##### See also:

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

#### 5.4.2.19 `cuDBGGetAPI::readRegister` [inherited]

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

**See also:**

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

**5.4.2.20 cudbgGetAPI::readReturnAddress** [inherited]

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

**See also:**

[readCallDepth](#)  
[readVirtualReturnAddress](#)

**5.4.2.21 cudbgGetAPI::readReturnAddress32** [inherited]

Reads the physical return address for a call level. Deprecated in 4.0.

**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

**See also:**

[readCallDepth32](#)

[readVirtualReturnAddress32](#)

**5.4.2.22 cudbgGetAPI::readSharedMemory** [inherited]

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

**See also:**

[readCodeMemory](#)

[readConstMemory](#)

[readGlobalMemory](#)

[readParamMemory](#)

[readLocalMemory](#)

[readTextureMemory](#)

[readRegister](#)

[readPC](#)

#### 5.4.2.23 `cudaGetAPI::readSyscallCallDepth` [inherited]

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

##### See also:

[readReturnAddress](#)  
[readVirtualReturnAddress](#)

#### 5.4.2.24 `cudaGetAPI::readTextureMemory` [inherited]

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

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

On sm\_30 and higher, use `readTextureMemoryBindless` instead.

##### 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

**See also:**

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

**5.4.2.25 cudbgGetAPI::readTextureMemoryBindless** [inherited]

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

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

For sm\_20 and lower, use readTextureMemory instead.

**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

**See also:**

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

#### 5.4.2.26 `cudbgGetAPI::readThreadId` [inherited]

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

##### See also:

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

#### 5.4.2.27 `cudbgGetAPI::readValidLanes` [inherited]

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

**See also:**

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

**5.4.2.28 cudbgGetAPI::readValidWarps** [inherited]

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

**See also:**

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

**5.4.2.29 cudbgGetAPI::readVirtualPC** [inherited]

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
```

**See also:**

[readPC](#)

**5.4.2.30 cudbgGetAPI::readVirtualReturnAddress** [inherited]

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
```

**See also:**

[readCallDepth](#)  
[readReturnAddress](#)

**5.4.2.31 cudbgGetAPI::readVirtualReturnAddress32** [inherited]

Reads the virtual return address for a call level. Deprecated in 4.0.

**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

**See also:**

[readCallDepth32](#)  
[readReturnAddress32](#)

#### 5.4.2.32 cudbgGetAPI::writePinnedMemory [inherited]

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

**See also:**

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

## 5.5 Device State Alteration

### Variables

- `CUDBGResult(* CUDBGAPI_st::writeGlobalMemory )(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+).*
- `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. Deprecated in 3.2.*
- `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.*
- `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.*
- `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.*
- `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.*

### 5.5.1 Detailed Description

### 5.5.2 Variable Documentation

#### 5.5.2.1 `cudbgGetAPI::writeGlobalMemory` [inherited]

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
```

**See also:**

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

**5.5.2.2 cudbgGetAPI::writeGlobalMemory31** [inherited]

Writes content to address in the global memory segment. Deprecated in 3.2.

**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
```

**See also:**

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

**5.5.2.3 cudbgGetAPI::writeLocalMemory** [inherited]

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

**See also:**

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

#### 5.5.2.4 `cudbgGetAPI::writeParamMemory` [inherited]

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

**See also:**

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

### 5.5.2.5 cudbgGetAPI::writeRegister [inherited]

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

#### See also:

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

### 5.5.2.6 cudbgGetAPI::writeSharedMemory [inherited]

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

**See also:**

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

## 5.6 Grid Properties

### Variables

- `CUDBGResult(* CUDBGAPI_st::getBlockDim )(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockDim)`  
*Get the number of threads in the given block.*
- `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.*
- `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. Deprecated in 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.*
- `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.*
- `CUDBGResult(* CUDBGAPI_st::getGridDim )(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *gridDim)`  
*Get the number of blocks in the given grid.*
- `CUDBGResult(* CUDBGAPI_st::getGridDim32 )(uint32_t dev, uint32_t sm, uint32_t wp, CuDim2 *gridDim)`  
*Get the number of blocks in the given grid. Deprecated in 4.0.*
- `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.*

### 5.6.1 Detailed Description

### 5.6.2 Variable Documentation

#### 5.6.2.1 `cudbgGetAPI::getBlockDim` [inherited]

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

**See also:**

[getGridDim](#)

**5.6.2.2 cudbgGetAPI::getElfImage** [inherited]

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

**5.6.2.3 cudbgGetAPI::getElfImage32** [inherited]

Get the relocated or non-relocated ELF image and size for the grid on the given device. Deprecated in 4.0.

**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

#### 5.6.2.4 `cudbgGetAPI::getGridAttribute` [inherited]

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

#### 5.6.2.5 `cudbgGetAPI::getGridAttributes` [inherited]

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

#### 5.6.2.6 `cudbgGetAPI::getGridDim` [inherited]

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

**See also:**

[getBlockDim](#)

**5.6.2.7 cudbgGetAPI::getGridDim32** [inherited]

Get the number of blocks in the given grid. Deprecated in 4.0.

**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

**See also:**

[getBlockDim](#)

**5.6.2.8 cudbgGetAPI::getTID** [inherited]

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

## 5.7 Device Properties

### Variables

- `CUDBGResult(* CUDBGAPI_st::getDeviceType)(uint32_t dev, char *buf, uint32_t sz)`  
*Get the string description of the device.*
- `CUDBGResult(* CUDBGAPI_st::getNumDevices)(uint32_t *numDev)`  
*Get the number of installed CUDA devices.*
- `CUDBGResult(* CUDBGAPI_st::getNumLanes)(uint32_t dev, uint32_t *numLanes)`  
*Get the number of lanes per warp on the device.*
- `CUDBGResult(* CUDBGAPI_st::getNumRegisters)(uint32_t dev, uint32_t *numRegs)`  
*Get the number of registers per lane on the device.*
- `CUDBGResult(* CUDBGAPI_st::getNumSMs)(uint32_t dev, uint32_t *numSMs)`  
*Get the total number of SMs on the device.*
- `CUDBGResult(* CUDBGAPI_st::getNumWarps)(uint32_t dev, uint32_t *numWarps)`  
*Get the number of warps per SM on the device.*
- `CUDBGResult(* CUDBGAPI_st::getSmType)(uint32_t dev, char *buf, uint32_t sz)`  
*Get the SM type of the device.*

### 5.7.1 Detailed Description

### 5.7.2 Variable Documentation

#### 5.7.2.1 `cudbgGetAPI::getDeviceType` [inherited]

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`

#### See also:

`getSMType`

### 5.7.2.2 `cudbgGetAPI::getNumDevices` [inherited]

Get the number of installed CUDA devices.

**Parameters:**

*numDev* - the returned number of devices

**Returns:**

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

**See also:**

[getNumSMs](#)  
[getNumWarps](#)  
[getNumLanes](#)  
[getNumRegisters](#)

### 5.7.2.3 `cudbgGetAPI::getNumLanes` [inherited]

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

**See also:**

[getNumDevices](#)  
[getNumSMs](#)  
[getNumWarps](#)  
[getNumRegisters](#)

### 5.7.2.4 `cudbgGetAPI::getNumRegisters` [inherited]

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

**See also:**

[getNumDevices](#)  
[getNumSMs](#)  
[getNumWarps](#)  
[getNumLanes](#)

**5.7.2.5 cudbgGetAPI::getNumSMs** [inherited]

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

**See also:**

[getNumDevices](#)  
[getNumWarps](#)  
[getNumLanes](#)  
[getNumRegisters](#)

**5.7.2.6 cudbgGetAPI::getNumWarps** [inherited]

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

**See also:**

[getNumDevices](#)  
[getNumSMs](#)  
[getNumLanes](#)  
[getNumRegisters](#)

**5.7.2.7 cudbgGetAPI::getSmType** [inherited]

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

**See also:**

[getDeviceType](#)

## 5.8 DWARF Utilities

### Variables

- `CUDBGResult(* CUDBGAPI_st::disassemble )(uint32_t dev, uint64_t addr, uint32_t *instSize, char *buf, uint32_t sz)`  
*Disassemble instruction at instruction address.*
- `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.*
- `CUDBGResult(* CUDBGAPI_st::getPhysicalRegister30 )(uint64_t pc, char *reg, uint32_t *buf, uint32_t sz, uint32_t *numPhysRegs, CUDBGRegClass *regClass)`  
*(DEPRECATED) Get the physical register number(s) assigned to a virtual register name 'reg' at a given PC, if 'reg' is live at that PC. The function has been deprecated. use getWarpPhysicalRegister instead.*
- `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.*
- `CUDBGResult(* CUDBGAPI_st::isDeviceCodeAddress )(uintptr_t addr, bool *isDeviceAddress)`  
*Determines whether a virtual address resides within device code.*
- `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.*

### 5.8.1 Detailed Description

### 5.8.2 Variable Documentation

#### 5.8.2.1 `cudbgGetAPI::disassemble` [inherited]

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

### 5.8.2.2 `cudbgGetAPI::getHostAddrFromDeviceAddr` [inherited]

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

#### See also:

[readGlobalMemory](#)  
[writeGlobalMemory](#)

### 5.8.2.3 `cudbgGetAPI::getPhysicalRegister30` [inherited]

(DEPRECATED) Get the physical register number(s) assigned to a virtual register name 'reg' at a given PC, if 'reg' is live at that PC. The function has been deprecated. use `getWarpPhysicalRegister` instead.

#### 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

### 5.8.2.4 `cudbgGetAPI::getPhysicalRegister40` [inherited]

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

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 the physical register with the highest physical register index will contain the lowest bits.

**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

**5.8.2.5 cudbgGetAPI::isDeviceCodeAddress** [inherited]

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

**5.8.2.6 cudbgGetAPI::lookupDeviceCodeSymbol** [inherited]

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

## 5.9 Events

### Data Structures

- struct [CUDBGEvent](#)  
*Event information container.*
- struct [CUDBGEvent30](#)  
*Event information container. Deprecated in 3.1.*
- struct [CUDBGEvent32](#)  
*Event information container. Deprecated in 4.0.*
- struct [CUDBGEvent42](#)  
*Event information container. Deprecated in 4.2.*
- struct [CUDBGEventCallbackData](#)  
*Event information passed to callback set with `setNotifyNewEventCallback` function.*
- struct [CUDBGEventCallbackData40](#)  
*Event information passed to callback set with `setNotifyNewEventCallback` function. Deprecated in 4.1.*

### Typedefs

- typedef void(\* [CUDBGNotifyNewEventCallback](#) )(CUDBGEventCallbackData \*data)  
*function type of the function called to notify debugger of the presence of a new event in the event queue.*
- typedef void(\* [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 3.2.*

### Enumerations

- enum [CUDBGEventKind](#) {  
[CUDBG\\_EVENT\\_INVALID](#),  
[CUDBG\\_EVENT\\_ELF\\_IMAGE\\_LOADED](#),  
[CUDBG\\_EVENT\\_KERNEL\\_READY](#),  
[CUDBG\\_EVENT\\_KERNEL\\_FINISHED](#),  
[CUDBG\\_EVENT\\_INTERNAL\\_ERROR](#),  
[CUDBG\\_EVENT\\_CTX\\_PUSH](#),  
[CUDBG\\_EVENT\\_CTX\\_POP](#),  
[CUDBG\\_EVENT\\_CTX\\_CREATE](#),  
[CUDBG\\_EVENT\\_CTX\\_DESTROY](#),  
[CUDBG\\_EVENT\\_TIMEOUT](#),  
[CUDBG\\_EVENT\\_ATTACH\\_COMPLETE](#) }  
*CUDA Kernel Events.*

## Variables

- [CUDBGResult\(\\* CUDBGAPI\\_st::acknowledgeEvent30 \)\(CUDBGEvent30 \\*event\)](#)  
*Inform the debugger API that the event has been processed. Deprecated in 3.1.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::acknowledgeEvents42 \)\(void\)](#)  
*Inform the debugger API that synchronous events have been processed. Deprecated in 5.0.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::acknowledgeSyncEvents \)\(void\)](#)  
*Inform the debugger API that synchronous events have been processed.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::getNextAsyncEvent \)\(CUDBGEvent \\*event\)](#)  
*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.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::getNextEvent30 \)\(CUDBGEvent30 \\*event\)](#)  
*Copies the next available event in the event queue into 'event' and removes it from the queue. Deprecated in 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. Deprecated in 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. Deprecated in 5.0.*
- [CUDBGResult\(\\* CUDBGAPI\\_st::getNextSyncEvent \)\(CUDBGEvent \\*event\)](#)  
*Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.*
- [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.*
- [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. Deprecated in 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. Deprecated in 4.1.*

### 5.9.1 Detailed Description

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](#).

## 5.9.2 Enumeration Type Documentation

### 5.9.2.1 enum CUDBGEventKind

CUDA Kernel Events.

**Enumerator:**

***CUDBG\_EVENT\_INVALID*** Invalid event.

***CUDBG\_EVENT\_ELF\_IMAGE\_LOADED*** The ELF image for a CUDA source module is available.

***CUDBG\_EVENT\_KERNEL\_READY*** A CUDA kernel is about to be launched.

***CUDBG\_EVENT\_KERNEL\_FINISHED*** A CUDA kernel has terminated.

***CUDBG\_EVENT\_INTERNAL\_ERROR*** An internal error occur. The debugging framework may be unstable.

***CUDBG\_EVENT\_CTX\_PUSH*** A CUDA context was pushed.

***CUDBG\_EVENT\_CTX\_POP*** A CUDA CTX was popped.

***CUDBG\_EVENT\_CTX\_CREATE*** A CUDA CTX was created.

***CUDBG\_EVENT\_CTX\_DESTROY*** A CUDA context was destroyed.

***CUDBG\_EVENT\_TIMEOUT*** An timeout event is sent at regular interval. This event can safely ge ignored.

***CUDBG\_EVENT\_ATTACH\_COMPLETE*** The attach process has completed and debugging of device code may start.

### 5.9.3 Variable Documentation

#### 5.9.3.1 ***cudbgGetAPI::acknowledgeEvent30*** [inherited]

Inform the debugger API that the event has been processed. Deprecated in 3.1.

**Parameters:**

*event* - pointer to the event that has been processed

**Returns:**

CUDBG\_SUCCESS

#### 5.9.3.2 ***cudbgGetAPI::acknowledgeEvents42*** [inherited]

Inform the debugger API that synchronous events have been processed. Deprecated in 5.0.

**Returns:**

CUDBG\_SUCCESS

#### 5.9.3.3 ***cudbgGetAPI::acknowledgeSyncEvents*** [inherited]

Inform the debugger API that synchronous events have been processed.

**Returns:**

CUDBG\_SUCCESS

#### 5.9.3.4 ***cudbgGetAPI::getNextAsyncEvent*** [inherited]

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

### 5.9.3.5 `cudbgGetAPI::getNextEvent30` [inherited]

Copies the next available event in the event queue into 'event' and removes it from the queue. Deprecated in 3.1.

#### 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

### 5.9.3.6 `cudbgGetAPI::getNextEvent32` [inherited]

Copies the next available event in the event queue into 'event' and removes it from the queue. Deprecated in 4.0.

#### 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

### 5.9.3.7 `cudbgGetAPI::getNextEvent42` [inherited]

Copies the next available event in the event queue into 'event' and removes it from the queue. Deprecated in 5.0.

#### 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

### 5.9.3.8 `cudbgGetAPI::getNextSyncEvent` [inherited]

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

### 5.9.3.9 cudbgGetAPI::setNotifyNewEventCallback [inherited]

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

### 5.9.3.10 cudbgGetAPI::setNotifyNewEventCallback31 [inherited]

Provides the API with the function to call to notify the debugger of a new application or device event. Deprecated in 3.2.

**Parameters:**

*callback* - the callback function

*data* - a pointer to be passed to the callback when called

**Returns:**

CUDBG\_SUCCESS

### 5.9.3.11 cudbgGetAPI::setNotifyNewEventCallback40 [inherited]

Provides the API with the function to call to notify the debugger of a new application or device event. Deprecated in 4.1.

**Parameters:**

*callback* - the callback function

**Returns:**

CUDBG\_SUCCESS

# Chapter 6

## Data Structure Documentation

### 6.1 cudbgGetAPI Struct Reference

The CUDA debugger API routines.

#### Data Fields

- [CUDBGResult\(\\* acknowledgeEvent30\)](#)(CUDBGEvent30 \*event)  
*Inform the debugger API that the event has been processed. Deprecated in 3.1.*
- [CUDBGResult\(\\* acknowledgeEvents42\)](#)(void)  
*Inform the debugger API that synchronous events have been processed. Deprecated in 5.0.*
- [CUDBGResult\(\\* acknowledgeSyncEvents\)](#)(void)  
*Inform the debugger API that synchronous events have been processed.*
- [CUDBGResult\(\\* clearAttachState\)](#)(void)  
*Clear attach-specific state prior to detach.*
- [CUDBGResult\(\\* disassemble\)](#)(uint32\_t dev, uint64\_t addr, uint32\_t \*instSize, char \*buf, uint32\_t sz)  
*Disassemble instruction at instruction address.*
- [CUDBGResult\(\\* finalize\)](#)(void)  
*Finalize the API and free all memory.*
- [CUDBGResult\(\\* getBlockDim\)](#)(uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim3 \*blockDim)  
*Get the number of threads in the given block.*
- [CUDBGResult\(\\* getDeviceType\)](#)(uint32\_t dev, char \*buf, uint32\_t sz)  
*Get the string description of the device.*
- [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.*

- `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. Deprecated in 4.0.*
- `CUDBGResult(* getGridAttribute )(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGAttribute attr, uint64_t *value)`  
*Get the value of a grid attribute.*
- `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.*
- `CUDBGResult(* getGridDim )(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *gridDim)`  
*Get the number of blocks in the given grid.*
- `CUDBGResult(* getGridDim32 )(uint32_t dev, uint32_t sm, uint32_t wp, CuDim2 *gridDim)`  
*Get the number of blocks in the given grid. Deprecated in 4.0.*
- `CUDBGResult(* getGridStatus )(uint32_t dev, uint32_t gridId, CUDBGGridStatus *status)`  
*Check whether the grid corresponding to the given gridId is still present on the device.*
- `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.*
- `CUDBGResult(* getNextAsyncEvent )(CUDBGEvent *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.*
- `CUDBGResult(* getNextEvent30 )(CUDBGEvent30 *event)`  
*Copies the next available event in the event queue into 'event' and removes it from the queue. Deprecated in 3.1.*
- `CUDBGResult(* getNextEvent32 )(CUDBGEvent32 *event)`  
*Copies the next available event in the event queue into 'event' and removes it from the queue. Deprecated in 4.0.*
- `CUDBGResult(* getNextEvent42 )(CUDBGEvent42 *event)`  
*Copies the next available event in the event queue into 'event' and removes it from the queue. Deprecated in 5.0.*
- `CUDBGResult(* getNextSyncEvent )(CUDBGEvent *event)`  
*Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.*
- `CUDBGResult(* getNumDevices )(uint32_t *numDev)`  
*Get the number of installed CUDA devices.*
- `CUDBGResult(* getNumLanes )(uint32_t dev, uint32_t *numLanes)`  
*Get the number of lanes per warp on the device.*
- `CUDBGResult(* getNumRegisters )(uint32_t dev, uint32_t *numRegs)`  
*Get the number of registers per lane on the device.*
- `CUDBGResult(* getNumSMs )(uint32_t dev, uint32_t *numSMs)`

*Get the total number of SMs on the device.*

- [CUDBGResult\(\\* getNumWarps\)](#) (uint32\_t dev, uint32\_t \*numWarps)  
*Get the number of warps per SM on the device.*
- [CUDBGResult\(\\* getPhysicalRegister30\)](#) (uint64\_t pc, char \*reg, uint32\_t \*buf, uint32\_t sz, uint32\_t \*numPhysRegs, CUDBGRegClass \*regClass)  
*(DEPRECATED) Get the physical register number(s) assigned to a virtual register name 'reg' at a given PC, if 'reg' is live at that PC. The function has been deprecated. use getWarpPhysicalRegister instead.*
- [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.*
- [CUDBGResult\(\\* getSmType\)](#) (uint32\_t dev, char \*buf, uint32\_t sz)  
*Get the SM type of the device.*
- [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.*
- [CUDBGResult\(\\* initialize\)](#) (void)  
*Initialize the API.*
- [CUDBGResult\(\\* isDeviceCodeAddress\)](#) (uintptr\_t addr, bool \*isDeviceAddress)  
*Determines whether a virtual address resides within device code.*
- [CUDBGResult\(\\* lookupDeviceCodeSymbol\)](#) (char \*symName, bool \*symFound, uintptr\_t \*symAddr)  
*Determines whether a symbol represents a function in device code and returns its virtual address.*
- [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.*
- [CUDBGResult\(\\* readActiveLanes\)](#) (uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*activeLanesMask)  
*Reads the bitmask of active lanes on a valid warp.*
- [CUDBGResult\(\\* readBlockIdx\)](#) (uint32\_t dev, uint32\_t sm, uint32\_t wp, CuDim3 \*blockIdx)  
*Reads the CUDA block index running on a valid warp.*
- [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. Deprecated in 4.0.*
- [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.*
- [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.*
- [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. Deprecated in 4.0.*

- [CUDBGResult\(\\* readCodeMemory\)](#)(uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)  
*Reads content at address in the code memory segment.*
- [CUDBGResult\(\\* readConstMemory\)](#)(uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)  
*Reads content at address in the constant memory segment.*
- [CUDBGResult\(\\* readGlobalMemory\)](#)(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+).*
- [CUDBGResult\(\\* readGlobalMemory31\)](#)(uint32\_t dev, uint64\_t addr, void \*buf, uint32\_t sz)  
*Reads content at address in the global memory segment. Deprecated in 3.2.*
- [CUDBGResult\(\\* readGridId\)](#)(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*gridId)  
*Reads the CUDA grid index running on a valid warp.*
- [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.*
- [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.*
- [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.*
- [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.*
- [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.*
- [CUDBGResult\(\\* readPinnedMemory\)](#)(uint64\_t addr, void \*buf, uint32\_t sz)  
*Reads content at pinned address in system memory.*
- [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.*
- [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.*
- [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. Deprecated in 4.0.*
- [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.*

- [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.*
- [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.*
- [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.*
- [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.*
- [CUDBGResult\(\\* readValidLanes\)](#)(uint32\_t dev, uint32\_t sm, uint32\_t wp, uint32\_t \*validLanesMask)  
*Reads the bitmask of valid lanes on a given warp.*
- [CUDBGResult\(\\* readValidWarps\)](#)(uint32\_t dev, uint32\_t sm, uint64\_t \*validWarpsMask)  
*Reads the bitmask of valid warps on a given SM.*
- [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.*
- [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.*
- [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. Deprecated in 4.0.*
- [CUDBGResult\(\\* requestCleanupOnDetach\)](#)(void)  
*Request for cleanup of driver state when detaching.*
- [CUDBGResult\(\\* resumeDevice\)](#)(uint32\_t dev)  
*Resume a suspended CUDA device.*
- [CUDBGResult\(\\* setBreakpoint\)](#)(uint32\_t dev, uint64\_t addr)  
*Sets a breakpoint at the given instruction address for the given device.*
- [CUDBGResult\(\\* setBreakpoint31\)](#)(uint64\_t addr)  
*Sets a breakpoint at the given instruction address. Deprecated in 3.2.*
- [CUDBGResult\(\\* setNotifyNewEventCallback\)](#)(CUDBGNotifyNewEventCallback callback)  
*Provides the API with the function to call to notify the debugger of a new application or device event.*
- [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. Deprecated in 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. Deprecated in 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.*
- [CUDBGResult\(\\* singleStepWarp40\)](#) (uint32\_t dev, uint32\_t sm, uint32\_t wp)  
*(DEPRECATED)Single step an individual warp on a suspended CUDA device. This function has been deprecated. Use [singleStepWarp\(\)](#) instead.*
- [CUDBGResult\(\\* suspendDevice\)](#) (uint32\_t dev)  
*Suspends a running CUDA device.*
- [CUDBGResult\(\\* unsetBreakpoint\)](#) (uint32\_t dev, uint64\_t addr)  
*Unsets a breakpoint at the given instruction address for the given device.*
- [CUDBGResult\(\\* unsetBreakpoint31\)](#) (uint64\_t addr)  
*Unsets a breakpoint at the given instruction address. Deprecated in 3.2.*
- [CUDBGResult\(\\* writeGlobalMemory\)](#) (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+).*
- [CUDBGResult\(\\* writeGlobalMemory31\)](#) (uint32\_t dev, uint64\_t addr, const void \*buf, uint32\_t sz)  
*Writes content to address in the global memory segment. Deprecated in 3.2.*
- [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.*
- [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.*
- [CUDBGResult\(\\* writePinnedMemory\)](#) (uint64\_t addr, const void \*buf, uint32\_t sz)  
*Writes content to pinned address in system memory.*
- [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.*
- [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.*

### 6.1.1 Detailed Description

The CUDA debugger API routines.

## 6.1.2 Field Documentation

### 6.1.2.1 cudbgGetAPI::clearAttachState

Clear attach-specific state prior to detach.

**Returns:**

CUDBG\_SUCCESS

### 6.1.2.2 cudbgGetAPI::getGridStatus

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

**Parameters:**

*devId* - device index

*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

### 6.1.2.3 cudbgGetAPI::requestCleanupOnDetach

Request for cleanup of driver state when detaching.

**Returns:**

CUDBG\_SUCCESS

## 6.2 CUDBGEvent Struct Reference

Event information container.

### Data Structures

- union [cases\\_st](#)

### Data Fields

- union [CUDBGEvent::cases\\_st cases](#)  
*Information for each type of event.*
- [CUDBGEventKind kind](#)  
*Event type.*

### 6.2.1 Detailed Description

Event information container.

## 6.3 CUDBGEvent30 Struct Reference

Event information container. Deprecated in 3.1.

### Data Structures

- union [cases30\\_st](#)

### Data Fields

- union [CUDBGEvent30::cases30\\_st](#) *cases*  
*Information for each type of event.*
- [CUDBGEventKind](#) *kind*  
*Event type.*

### 6.3.1 Detailed Description

Event information container. Deprecated in 3.1.

## 6.4 CUDBGEvent30::CUDBGEvent30::cases30\_st Union Reference

### Data Structures

- struct [elfImageLoaded30\\_st](#)
- struct [kernelFinished30\\_st](#)
- struct [kernelReady30\\_st](#)

### Data Fields

- struct [CUDBGEvent30::cases30\\_st::elfImageLoaded30\\_st](#) `elfImageLoaded`  
*Information about the loaded ELF image.*
- struct [CUDBGEvent30::cases30\\_st::kernelFinished30\\_st](#) `kernelFinished`  
*Information about the kernel that just terminated.*
- struct [CUDBGEvent30::cases30\\_st::kernelReady30\\_st](#) `kernelReady`  
*Information about the kernel ready to be launched.*

### 6.4.1 Detailed Description

## 6.5 CUDBGEvent30::CUDBGEvent30::cases30\_st::CUDBGEvent30::cases30\_st::elfImageLoaded30\_st Struct Reference

### Data Fields

- char \* [nonRelocatedElfImage](#)  
*pointer to the non-relocated ELF image for a CUDA source module.*
- char \* [relocatedElfImage](#)  
*pointer to the relocated ELF image for a CUDA source module.*
- uint32\_t [size](#)  
*size of the ELF image (32-bit).*

### 6.5.1 Detailed Description

## 6.6 CUDBGEvent30::CUDBGEvent30::cases30\_st::CUDBGEvent30::cases30\_st::kernelFinished30\_st Struct Reference

### Data Fields

- uint32\_t [dev](#)  
*device index of the kernel.*
- uint32\_t [gridId](#)  
*grid index of the kernel.*
- uint32\_t [tid](#)  
*host thread id (or LWP id) of the thread hosting the kernel (Linux only).*

### 6.6.1 Detailed Description

## 6.7 CUDBGEvent30::CUDBGEvent30::cases30\_st::CUDBGEvent30::cases30\_st::kernelReady30\_st Struct Reference

### Data Fields

- uint32\_t [dev](#)  
*device index of the kernel.*
- uint32\_t [gridId](#)  
*grid index of the kernel.*
- uint32\_t [tid](#)  
*host thread id (or LWP id) of the thread hosting the kernel (Linux only).*

### 6.7.1 Detailed Description

## 6.8 CUDBGEvent32 Struct Reference

Event information container. Deprecated in 4.0.

### Data Structures

- union [cases32\\_st](#)

### Data Fields

- union [CUDBGEvent32::cases32\\_st](#) *cases*  
*Information for each type of event.*
- [CUDBGEventKind](#) *kind*  
*Event type.*

### 6.8.1 Detailed Description

Event information container. Deprecated in 4.0.

## 6.9 CUDBGEvent32::CUDBGEvent32::cases32\_st Union Reference

### Data Structures

- struct [contextCreate32\\_st](#)
- struct [contextDestroy32\\_st](#)
- struct [contextPop32\\_st](#)
- struct [contextPush32\\_st](#)
- struct [elfImageLoaded32\\_st](#)
- struct [kernelFinished32\\_st](#)
- struct [kernelReady32\\_st](#)

### Data Fields

- struct [CUDBGEvent32::cases32\\_st::contextCreate32\\_st](#) [contextCreate](#)  
*Information about the context being created.*
- struct [CUDBGEvent32::cases32\\_st::contextDestroy32\\_st](#) [contextDestroy](#)  
*Information about the context being destroyed.*
- struct [CUDBGEvent32::cases32\\_st::contextPop32\\_st](#) [contextPop](#)  
*Information about the context being popped.*
- struct [CUDBGEvent32::cases32\\_st::contextPush32\\_st](#) [contextPush](#)  
*Information about the context being pushed.*
- struct [CUDBGEvent32::cases32\\_st::elfImageLoaded32\\_st](#) [elfImageLoaded](#)  
*Information about the loaded ELF image.*
- struct [CUDBGEvent32::cases32\\_st::kernelFinished32\\_st](#) [kernelFinished](#)  
*Information about the kernel that just terminated.*
- struct [CUDBGEvent32::cases32\\_st::kernelReady32\\_st](#) [kernelReady](#)  
*Information about the kernel ready to be launched.*

### 6.9.1 Detailed Description

## 6.10 CUDBGEvent32::CUDBGEvent32::cases32\_st::CUDBGEvent32::cases32\_st::contextCreate32\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being created.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.10.1 Detailed Description

## 6.11 CUDBGEvent32::CUDBGEvent32::cases32\_st::CUDBGEvent32::cases32\_st::contextDestroy32\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being destroyed.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.11.1 Detailed Description

## 6.12 CUDBGEvent32::CUDBGEvent32::cases32\_st::CUDBGEvent32::cases32\_st::contextPop32\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being popped.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.12.1 Detailed Description

## 6.13 CUDBGEvent32::CUDBGEvent32::cases32\_st::CUDBGEvent32::cases32\_st::contextPush32\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being pushed.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.13.1 Detailed Description

## 6.14 CUDBGEvent32::CUDBGEvent32::cases32\_st::CUDBGEvent32::cases32\_st::elfImageLoaded32\_st Struct Reference

### Data Fields

- `uint64_t context`  
*context of the kernel.*
- `uint32_t dev`  
*device index of the kernel.*
- `uint64_t module`  
*module of the kernel.*
- `char * nonRelocatedElfImage`  
*pointer to the non-relocated ELF image for a CUDA source module.*
- `char * relocatedElfImage`  
*pointer to the relocated ELF image for a CUDA source module.*
- `uint32_t size`  
*size of the ELF image (32-bit).*

### 6.14.1 Detailed Description

## 6.15 CUDBGEvent32::CUDBGEvent32::cases32\_st::CUDBGEvent32::cases32\_st::kernelFinished32\_st Struct Reference

### Data Fields

- `uint64_t context`  
*context of the kernel.*
- `uint32_t dev`  
*device index of the kernel.*
- `uint64_t function`  
*function of the kernel.*
- `uint64_t functionEntry`  
*entry PC of the kernel.*
- `uint32_t gridId`  
*grid index of the kernel.*
- `uint64_t module`  
*module of the kernel.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the kernel (Linux only).*

### 6.15.1 Detailed Description

## 6.16 CUDBGEvent32::CUDBGEvent32::cases32\_st::CUDBGEvent32::cases32\_st::kernelReady32\_st Struct Reference

### Data Fields

- `uint64_t context`  
*context of the kernel.*
- `uint32_t dev`  
*device index of the kernel.*
- `uint64_t function`  
*function of the kernel.*
- `uint64_t functionEntry`  
*entry PC of the kernel.*
- `uint32_t gridId`  
*grid index of the kernel.*
- `uint64_t module`  
*module of the kernel.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the kernel (Linux only).*

### 6.16.1 Detailed Description

## 6.17 CUDBGEvent42 Struct Reference

Event information container. Deprecated in 4.2.

### Data Structures

- union [cases42\\_st](#)

### Data Fields

- union [CUDBGEvent42::cases42\\_st](#) *cases*  
*Information for each type of event.*
- [CUDBGEventKind](#) *kind*  
*Event type.*

### 6.17.1 Detailed Description

Event information container. Deprecated in 4.2.

## 6.18 CUDBGEvent42::CUDBGEvent42::cases42\_st Union Reference

### Data Structures

- struct [contextCreate42\\_st](#)
- struct [contextDestroy42\\_st](#)
- struct [contextPop42\\_st](#)
- struct [contextPush42\\_st](#)
- struct [elfImageLoaded42\\_st](#)
- struct [kernelFinished42\\_st](#)
- struct [kernelReady42\\_st](#)

### Data Fields

- struct [CUDBGEvent42::cases42\\_st::contextCreate42\\_st](#) [contextCreate](#)  
*Information about the context being created.*
- struct [CUDBGEvent42::cases42\\_st::contextDestroy42\\_st](#) [contextDestroy](#)  
*Information about the context being destroyed.*
- struct [CUDBGEvent42::cases42\\_st::contextPop42\\_st](#) [contextPop](#)  
*Information about the context being popped.*
- struct [CUDBGEvent42::cases42\\_st::contextPush42\\_st](#) [contextPush](#)  
*Information about the context being pushed.*
- struct [CUDBGEvent42::cases42\\_st::elfImageLoaded42\\_st](#) [elfImageLoaded](#)  
*Information about the loaded ELF image.*
- struct [CUDBGEvent42::cases42\\_st::kernelFinished42\\_st](#) [kernelFinished](#)  
*Information about the kernel that just terminated.*
- struct [CUDBGEvent42::cases42\\_st::kernelReady42\\_st](#) [kernelReady](#)  
*Information about the kernel ready to be launched.*

### 6.18.1 Detailed Description

## 6.19 CUDBGEvent42::CUDBGEvent42::cases42\_st::CUDBGEvent42::cases42\_st::contextCreate42\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being created.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.19.1 Detailed Description

## 6.20 CUDBGEvent42::CUDBGEvent42::cases42\_st::CUDBGEvent42::cases42\_st::contextDestroy42\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being destroyed.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.20.1 Detailed Description

## 6.21 CUDBGEvent42::CUDBGEvent42::cases42\_st::CUDBGEvent42::cases42\_st::contextPop42\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being popped.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.21.1 Detailed Description

## 6.22 CUDBGEvent42::CUDBGEvent42::cases42\_st::CUDBGEvent42::cases42\_st::contextPush42\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being pushed.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.22.1 Detailed Description

## 6.23 CUDBGEvent42::CUDBGEvent42::cases42\_st::CUDBGEvent42::cases42\_st::elfImageLoaded42\_st Struct Reference

### Data Fields

- `uint64_t context`  
*context of the kernel.*
- `uint32_t dev`  
*device index of the kernel.*
- `uint64_t module`  
*module of the kernel.*
- `char * nonRelocatedElfImage`  
*pointer to the non-relocated ELF image for a CUDA source module.*
- `char * relocatedElfImage`  
*pointer to the relocated ELF image for a CUDA source module.*
- `uint64_t size`  
*size of the ELF image (64-bit).*
- `uint32_t size32`  
*size of the ELF image (32-bit). Deprecated in 4.0.*

### 6.23.1 Detailed Description

## 6.24 CUDBGEvent42::CUDBGEvent42::cases42\_st::CUDBGEvent42::cases42\_st::kernelFinished42\_st Struct Reference

### Data Fields

- `uint64_t context`  
*context of the kernel.*
- `uint32_t dev`  
*device index of the kernel.*
- `uint64_t function`  
*function of the kernel.*
- `uint64_t functionEntry`  
*entry PC of the kernel.*
- `uint32_t gridId`  
*grid index of the kernel.*
- `uint64_t module`  
*module of the kernel.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the kernel (Linux only).*

### 6.24.1 Detailed Description

## 6.25 CUDBGEvent42::CUDBGEvent42::cases42\_st::CUDBGEvent42::cases42\_st::kernelReady42\_st Struct Reference

### Data Fields

- CuDim3 [blockDim](#)  
*block dimensions of the kernel.*
- uint64\_t [context](#)  
*context of the kernel.*
- uint32\_t [dev](#)  
*device index of the kernel.*
- uint64\_t [function](#)  
*function of the kernel.*
- uint64\_t [functionEntry](#)  
*entry PC of the kernel.*
- CuDim3 [gridDim](#)  
*grid dimensions of the kernel.*
- uint32\_t [gridId](#)  
*grid index of the kernel.*
- uint64\_t [module](#)  
*module of the kernel.*
- uint32\_t [tid](#)  
*host thread id (or LWP id) of the thread hosting the kernel (Linux only).*
- [CUDBGKernelType](#) [type](#)  
*the type of the kernel: system or application.*

### 6.25.1 Detailed Description

## 6.26 CUDBGEvent::CUDBGEvent::cases\_st Union Reference

### Data Structures

- struct [contextCreate\\_st](#)
- struct [contextDestroy\\_st](#)
- struct [contextPop\\_st](#)
- struct [contextPush\\_st](#)
- struct [elfImageLoaded\\_st](#)
- struct [internalError\\_st](#)
- struct [kernelFinished\\_st](#)
- struct [kernelReady\\_st](#)

### Data Fields

- struct [CUDBGEvent::cases\\_st::contextCreate\\_st](#) contextCreate  
*Information about the context being created.*
- struct [CUDBGEvent::cases\\_st::contextDestroy\\_st](#) contextDestroy  
*Information about the context being destroyed.*
- struct [CUDBGEvent::cases\\_st::contextPop\\_st](#) contextPop  
*Information about the context being popped.*
- struct [CUDBGEvent::cases\\_st::contextPush\\_st](#) contextPush  
*Information about the context being pushed.*
- struct [CUDBGEvent::cases\\_st::elfImageLoaded\\_st](#) elfImageLoaded  
*Information about the loaded ELF image.*
- struct [CUDBGEvent::cases\\_st::internalError\\_st](#) internalError  
*Information about internal errors.*
- struct [CUDBGEvent::cases\\_st::kernelFinished\\_st](#) kernelFinished  
*Information about the kernel that just terminated.*
- struct [CUDBGEvent::cases\\_st::kernelReady\\_st](#) kernelReady  
*Information about the kernel ready to be launched.*

### 6.26.1 Detailed Description

## 6.27 CUDBGEvent::CUDBGEvent::cases\_st::CUDBGEvent::cases\_st::contextCreate\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being created.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.27.1 Detailed Description

## 6.28 CUDBGEvent::CUDBGEvent::cases\_st::CUDBGEvent::cases\_st::contextDestroy\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being destroyed.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.28.1 Detailed Description

## 6.29 CUDBGEvent::CUDBGEvent::cases\_st::CUDBGEvent::cases\_st::contextPop\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being popped.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.29.1 Detailed Description

## 6.30 CUDBGEvent::CUDBGEvent::cases\_st::CUDBGEvent::cases\_st::contextPush\_st Struct Reference

### Data Fields

- `uint64_t context`  
*the context being pushed.*
- `uint32_t dev`  
*device index of the context.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the context (Linux only).*

### 6.30.1 Detailed Description

## 6.31 CUDBGEvent::CUDBGEvent::cases\_st::CUDBGEvent::cases\_st::elfImageLoaded\_st Struct Reference

### Data Fields

- `uint64_t context`  
*context of the kernel.*
- `uint32_t dev`  
*device index of the kernel.*
- `uint64_t module`  
*module of the kernel.*
- `char * nonRelocatedElfImage`  
*pointer to the non-relocated ELF image for a CUDA source module.*
- `char * relocatedElfImage`  
*pointer to the relocated ELF image for a CUDA source module.*
- `uint64_t size`  
*size of the ELF image (64-bit).*
- `uint32_t size32`  
*size of the ELF image (32-bit). Deprecated in 4.0.*

### 6.31.1 Detailed Description

## 6.32 CUDBGEvent::CUDBGEvent::cases\_st::CUDBGEvent::cases\_st::internalError\_st Struct Reference

### Data Fields

- [CUDBGResult errorType](#)  
*Type of the internal error.*

### 6.32.1 Detailed Description

## 6.33 CUDBGEvent::CUDBGEvent::cases\_st::CUDBGEvent::cases\_st::kernelFinished\_st Struct Reference

### Data Fields

- `uint64_t context`  
*context of the kernel.*
- `uint32_t dev`  
*device index of the kernel.*
- `uint64_t function`  
*function of the kernel.*
- `uint64_t functionEntry`  
*entry PC of the kernel.*
- `uint32_t gridId`  
*grid index of the kernel.*
- `uint64_t module`  
*module of the kernel.*
- `uint32_t tid`  
*host thread id (or LWP id) of the thread hosting the kernel (Linux only).*

### 6.33.1 Detailed Description

## 6.34 CUDBGEvent::CUDBGEvent::cases\_st::CUDBGEvent::cases\_st::kernelReady\_st Struct Reference

### Data Fields

- CuDim3 [blockDim](#)  
*block dimensions of the kernel.*
- uint64\_t [context](#)  
*context of the kernel.*
- uint32\_t [dev](#)  
*device index of the kernel.*
- uint64\_t [function](#)  
*function of the kernel.*
- uint64\_t [functionEntry](#)  
*entry PC of the kernel.*
- CuDim3 [gridDim](#)  
*grid dimensions of the kernel.*
- uint32\_t [gridId](#)  
*grid index of the kernel.*
- uint64\_t [module](#)  
*module of the kernel.*
- uint32\_t [tid](#)  
*host thread id (or LWP id) of the thread hosting the kernel (Linux only).*
- [CUDBGKernelType](#) [type](#)  
*the type of the kernel: system or application.*

### 6.34.1 Detailed Description

## 6.35 CUDBGEventCallbackData Struct Reference

Event information passed to callback set with setNotifyNewEventCallback function.

### Data Fields

- uint32\_t [tid](#)

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

- uint32\_t [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.*

### 6.35.1 Detailed Description

Event information passed to callback set with setNotifyNewEventCallback function.

## 6.36 CUDBGEventCallbackData40 Struct Reference

Event information passed to callback set with setNotifyNewEventCallback function. Deprecated in 4.1.

### Data Fields

- `uint32_t tid`

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

### 6.36.1 Detailed Description

Event information passed to callback set with setNotifyNewEventCallback function. Deprecated in 4.1.

# Chapter 7

## File Documentation

### 7.1 cudadebugger.h File Reference

Header file for the CUDA debugger API.

#### Data Structures

- struct [cudbgGetAPI](#)  
*The CUDA debugger API routines.*
- struct [CUDBGEvent](#)  
*Event information container.*
- struct [CUDBGEvent30](#)  
*Event information container. Deprecated in 3.1.*
- union [CUDBGEvent30::CUDBGEvent30::cases30\\_st](#)
- struct [CUDBGEvent30::CUDBGEvent30::cases30\\_st::CUDBGEvent30::cases30\\_st::elfImageLoaded30\\_st](#)
- struct [CUDBGEvent30::CUDBGEvent30::cases30\\_st::CUDBGEvent30::cases30\\_st::kernelFinished30\\_st](#)
- struct [CUDBGEvent30::CUDBGEvent30::cases30\\_st::CUDBGEvent30::cases30\\_st::kernelReady30\\_st](#)
- struct [CUDBGEvent32](#)  
*Event information container. Deprecated in 4.0.*
- union [CUDBGEvent32::CUDBGEvent32::cases32\\_st](#)
- struct [CUDBGEvent32::CUDBGEvent32::cases32\\_st::CUDBGEvent32::cases32\\_st::contextCreate32\\_st](#)
- struct [CUDBGEvent32::CUDBGEvent32::cases32\\_st::CUDBGEvent32::cases32\\_st::contextDestroy32\\_st](#)
- struct [CUDBGEvent32::CUDBGEvent32::cases32\\_st::CUDBGEvent32::cases32\\_st::contextPop32\\_st](#)
- struct [CUDBGEvent32::CUDBGEvent32::cases32\\_st::CUDBGEvent32::cases32\\_st::contextPush32\\_st](#)
- struct [CUDBGEvent32::CUDBGEvent32::cases32\\_st::CUDBGEvent32::cases32\\_st::elfImageLoaded32\\_st](#)
- struct [CUDBGEvent32::CUDBGEvent32::cases32\\_st::CUDBGEvent32::cases32\\_st::kernelFinished32\\_st](#)
- struct [CUDBGEvent32::CUDBGEvent32::cases32\\_st::CUDBGEvent32::cases32\\_st::kernelReady32\\_st](#)
- struct [CUDBGEvent42](#)  
*Event information container. Deprecated in 4.2.*
- union [CUDBGEvent42::CUDBGEvent42::cases42\\_st](#)

- struct `CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextCreate42_st`
- struct `CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextDestroy42_st`
- struct `CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextPop42_st`
- struct `CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::contextPush42_st`
- struct `CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::elfImageLoaded42_st`
- struct `CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::kernelFinished42_st`
- struct `CUDBGEvent42::CUDBGEvent42::cases42_st::CUDBGEvent42::cases42_st::kernelReady42_st`
- union `CUDBGEvent::CUDBGEvent::cases_st`
- struct `CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextCreate_st`
- struct `CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextDestroy_st`
- struct `CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPop_st`
- struct `CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPush_st`
- struct `CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st`
- struct `CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::internalError_st`
- struct `CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelFinished_st`
- struct `CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st`
- struct `CUDBGEventCallbackData`

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

- struct `CUDBGEventCallbackData40`

*Event information passed to callback set with `setNotifyNewEventCallback` function. Deprecated in 4.1.*

## Typedefs

- typedef `void(* CUDBGNotifyNewEventCallback)(CUDBGEventCallbackData *data)`  
*function type of the function called to notify debugger of the presence of a new event in the event queue.*
- typedef `void(* 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 3.2.*

## Enumerations

- enum `CUDBGAttribute` {  
`CUDBG_ATTR_GRID_LAUNCH_BLOCKING,`  
`CUDBG_ATTR_GRID_TID` }  
*Query attribute.*
- enum `CUDBGEventKind` {  
`CUDBG_EVENT_INVALID,`  
`CUDBG_EVENT_ELF_IMAGE_LOADED,`  
`CUDBG_EVENT_KERNEL_READY,`  
`CUDBG_EVENT_KERNEL_FINISHED,`  
`CUDBG_EVENT_INTERNAL_ERROR,`  
`CUDBG_EVENT_CTX_PUSH,`  
`CUDBG_EVENT_CTX_POP,`

```
CUDBG_EVENT_CTX_CREATE,  
CUDBG_EVENT_CTX_DESTROY,  
CUDBG_EVENT_TIMEOUT,  
CUDBG_EVENT_ATTACH_COMPLETE }
```

*CUDA Kernel Events.*

- enum CUDBGException\_t {  
CUDBG\_EXCEPTION\_UNKNOWN,  
CUDBG\_EXCEPTION\_NONE,  
CUDBG\_EXCEPTION\_LANE\_ILLEGAL\_ADDRESS,  
CUDBG\_EXCEPTION\_LANE\_USER\_STACK\_OVERFLOW,  
CUDBG\_EXCEPTION\_DEVICE\_HARDWARE\_STACK\_OVERFLOW,  
CUDBG\_EXCEPTION\_WARP\_ILLEGAL\_INSTRUCTION,  
CUDBG\_EXCEPTION\_WARP\_OUT\_OF\_RANGE\_ADDRESS,  
CUDBG\_EXCEPTION\_WARP\_MISALIGNED\_ADDRESS,  
CUDBG\_EXCEPTION\_WARP\_INVALID\_ADDRESS\_SPACE,  
CUDBG\_EXCEPTION\_WARP\_INVALID\_PC,  
CUDBG\_EXCEPTION\_WARP\_HARDWARE\_STACK\_OVERFLOW,  
CUDBG\_EXCEPTION\_DEVICE\_ILLEGAL\_ADDRESS,  
CUDBG\_EXCEPTION\_LANE\_MISALIGNED\_ADDRESS }

*Hardware Exception Types.*

- enum CUDBGGridStatus {  
CUDBG\_GRID\_STATUS\_INVALID,  
CUDBG\_GRID\_STATUS\_PENDING,  
CUDBG\_GRID\_STATUS\_ACTIVE,  
CUDBG\_GRID\_STATUS\_SLEEPING,  
CUDBG\_GRID\_STATUS\_TERMINATED,  
CUDBG\_GRID\_STATUS\_UNDETERMINED }

*Grid status.*

- enum CUDBGKernelType {  
CUDBG\_KNL\_TYPE\_UNKNOWN,  
CUDBG\_KNL\_TYPE\_SYSTEM,  
CUDBG\_KNL\_TYPE\_APPLICATION }

*Kernel types.*

- enum CUDBGRegClass {  
REG\_CLASS\_INVALID,  
REG\_CLASS\_REG\_CC,  
REG\_CLASS\_REG\_PRED,  
REG\_CLASS\_REG\_ADDR,  
REG\_CLASS\_REG\_HALF,

```
REG_CLASS_REG_FULL,  
REG_CLASS_MEM_LOCAL,  
REG_CLASS_LMEM_REG_OFFSET }
```

*Physical register types.*

- enum CUDBGResult {  
CUDBG\_SUCCESS,  
CUDBG\_ERROR\_UNKNOWN,  
CUDBG\_ERROR\_BUFFER\_TOO\_SMALL,  
CUDBG\_ERROR\_UNKNOWN\_FUNCTION,  
CUDBG\_ERROR\_INVALID\_ARGS,  
CUDBG\_ERROR\_UNINITIALIZED,  
CUDBG\_ERROR\_INVALID\_COORDINATES,  
CUDBG\_ERROR\_INVALID\_MEMORY\_SEGMENT,  
CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS,  
CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED,  
CUDBG\_ERROR\_INTERNAL,  
CUDBG\_ERROR\_INVALID\_DEVICE,  
CUDBG\_ERROR\_INVALID\_SM,  
CUDBG\_ERROR\_INVALID\_WARP,  
CUDBG\_ERROR\_INVALID\_LANE,  
CUDBG\_ERROR\_SUSPENDED\_DEVICE,  
CUDBG\_ERROR\_RUNNING\_DEVICE,  
CUDBG\_ERROR\_INVALID\_ADDRESS,  
CUDBG\_ERROR\_INCOMPATIBLE\_API,  
CUDBG\_ERROR\_INITIALIZATION\_FAILURE,  
CUDBG\_ERROR\_INVALID\_GRID,  
CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE,  
CUDBG\_ERROR\_SOME\_DEVICES\_WATCHDOGGED,  
CUDBG\_ERROR\_ALL\_DEVICES\_WATCHDOGGED,  
CUDBG\_ERROR\_INVALID\_ATTRIBUTE,  
CUDBG\_ERROR\_ZERO\_CALL\_DEPTH,  
CUDBG\_ERROR\_INVALID\_CALL\_LEVEL,  
CUDBG\_ERROR\_COMMUNICATION\_FAILURE,  
CUDBG\_ERROR\_INVALID\_CONTEXT,  
CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM ,  
CUDBG\_ERROR\_OS\_RESOURCES,  
CUDBG\_ERROR\_FORK\_FAILED }

*Result values of all the API routines.*

## Functions

- **CUDBGResult cudbgGetAPIVersion** (uint32\_t \*major, uint32\_t \*minor, uint32\_t \*rev)  
Get the API version supported by the CUDA driver.

### 7.1.1 Detailed Description

Header file for the CUDA debugger API.

### 7.1.2 Enumeration Type Documentation

#### 7.1.2.1 enum CUDBGAttribute

Query attribute.

##### Enumerator:

- CUDBG\_ATTR\_GRID\_LAUNCH\_BLOCKING** whether the launch is synchronous or not.
- CUDBG\_ATTR\_GRID\_TID** The id of the host thread that launched the grid.

#### 7.1.2.2 enum CUDBGException\_t

Hardware Exception Types.

##### Enumerator:

- CUDBG\_EXCEPTION\_UNKNOWN** Reported if we do not know what exception the chip has hit (global error).
- CUDBG\_EXCEPTION\_NONE** Reported when there is no exception on the chip (no error).
- CUDBG\_EXCEPTION\_LANE\_ILLEGAL\_ADDRESS** Reported when memcheck(enabled within cuda-gdb) finds access violations (lane error: precise software generated exception).
- CUDBG\_EXCEPTION\_LANE\_USER\_STACK\_OVERFLOW** Reported from user (data) stack overflow checks in each function's prologue (lane error: precise software generated exception, ABI-only).
- CUDBG\_EXCEPTION\_DEVICE\_HARDWARE\_STACK\_OVERFLOW** Reported if CRS overflows (global error: the warp that caused this will terminate).
- CUDBG\_EXCEPTION\_WARP\_ILLEGAL\_INSTRUCTION** Reported when any lane in a warp executes an illegal instruction (warp error: invalid branch target, invalid opcode, misaligned/oor reg, invalid immediates, etc.).
- CUDBG\_EXCEPTION\_WARP\_OUT\_OF\_RANGE\_ADDRESS** Reported when any lane in a warp accesses memory that is out of range (warp error: lmem\_lo/hi, shared, and 40-bit va accesses).
- CUDBG\_EXCEPTION\_WARP\_MISALIGNED\_ADDRESS** Reported when any lane in a warp accesses memory that is misaligned (warp error: lmem\_lo/hi, shared, and 40-bit va accesses).
- CUDBG\_EXCEPTION\_WARP\_INVALID\_ADDRESS\_SPACE** Reported when any lane in a warp executes an instruction that accesses a memory space that is not permitted for that instruction (warp error).
- CUDBG\_EXCEPTION\_WARP\_INVALID\_PC** Reported when any lane in a warp advances its PC beyond the 32-bit address space (warp error).
- CUDBG\_EXCEPTION\_WARP\_HARDWARE\_STACK\_OVERFLOW** Reported when any lane in a warp hits (uncommon) stack issues (warp error: stack error or api stack overflow).

***CUDBG\_EXCEPTION\_DEVICE\_ILLEGAL\_ADDRESS*** Reported when MMU detects an error (global error: L1 error status field is set in the global esr – for the most part this catches errors SM couldn't catch with oor address detection).

***CUDBG\_EXCEPTION\_LANE\_MISALIGNED\_ADDRESS*** Reported when memcheck(enabled within cuda-gdb) finds access violations (lane error: precise software generated exception).

### 7.1.2.3 enum CUDBGGridStatus

Grid status.

#### Enumerator:

***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.

### 7.1.2.4 enum CUDBGKernelType

Kernel types.

#### Enumerator:

***CUDBG\_KNL\_TYPE\_UNKNOWN*** Unknown kernel type. Fall-back value.

***CUDBG\_KNL\_TYPE\_SYSTEM*** System kernel, launched by the CUDA driver (cudaMemset, ...).

***CUDBG\_KNL\_TYPE\_APPLICATION*** Application kernel, launched by the application.

### 7.1.2.5 enum CUDBGRegClass

Physical register types.

#### Enumerator:

***REG\_CLASS\_INVALID*** The physical register is invalid.

***REG\_CLASS\_REG\_CC*** The physical register is a condition code register. Unused.

***REG\_CLASS\_REG\_PRED*** The physical register is a predicate register. Unused.

***REG\_CLASS\_REG\_ADDR*** The physical register is an address register. Unused.

***REG\_CLASS\_REG\_HALF*** The physical register is a 16-bit register. Unused.

***REG\_CLASS\_REG\_FULL*** The physical register is a 32-bit register.

***REG\_CLASS\_MEM\_LOCAL*** The content of the physical register has been spilled to memory.

***REG\_CLASS\_LMEM\_REG\_OFFSET*** The content of the physical register has been spilled to the local stack (ABI only).

### 7.1.2.6 enum CUDBGResult

Result values of all the API routines.

#### Enumerator:

- CUDBG\_SUCCESS* The API call executed successfully.
- CUDBG\_ERROR\_UNKNOWN* Error type not listed below.
- CUDBG\_ERROR\_BUFFER\_TOO\_SMALL* Cannot copy all the queried data into the buffer argument.
- CUDBG\_ERROR\_UNKNOWN\_FUNCTION* Function cannot be found in the CUDA kernel.
- CUDBG\_ERROR\_INVALID\_ARGS* Wrong use of arguments (NULL pointer, illegal value,...).
- CUDBG\_ERROR\_UNINITIALIZED* Debugger API has not yet been properly initialized.
- CUDBG\_ERROR\_INVALID\_COORDINATES* Invalid block or thread coordinates were provided.
- CUDBG\_ERROR\_INVALID\_MEMORY\_SEGMENT* Invalid memory segment requested.
- CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS* Requested address (+size) is not within proper segment boundaries.
- CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED* Memory is not mapped and cannot be mapped.
- CUDBG\_ERROR\_INTERNAL* A debugger internal error occurred.
- CUDBG\_ERROR\_INVALID\_DEVICE* Specified device cannot be found.
- CUDBG\_ERROR\_INVALID\_SM* Specified sm cannot be found.
- CUDBG\_ERROR\_INVALID\_WARP* Specified warp cannot be found.
- CUDBG\_ERROR\_INVALID\_LANE* Specified lane cannot be found.
- CUDBG\_ERROR\_SUSPENDED\_DEVICE* The requested operation is not allowed when the device is suspended.
- CUDBG\_ERROR\_RUNNING\_DEVICE* Device is running and not suspended.
- CUDBG\_ERROR\_INVALID\_ADDRESS* Address is out-of-range.
- CUDBG\_ERROR\_INCOMPATIBLE\_API* The requested API is not available.
- CUDBG\_ERROR\_INITIALIZATION\_FAILURE* The API could not be initialized.
- CUDBG\_ERROR\_INVALID\_GRID* The specified grid is not valid.
- CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE* The event queue is empty and there is no event left to be processed.
- CUDBG\_ERROR\_SOME\_DEVICES\_WATCHDOGGED* Some devices were excluded because they have a watchdog associated with them.
- CUDBG\_ERROR\_ALL\_DEVICES\_WATCHDOGGED* All devices were excluded because they have a watchdog associated with them.
- CUDBG\_ERROR\_INVALID\_ATTRIBUTE* Specified attribute does not exist or is incorrect.
- CUDBG\_ERROR\_ZERO\_CALL\_DEPTH* No function calls have been made on the device.
- CUDBG\_ERROR\_INVALID\_CALL\_LEVEL* Specified call level is invalid.
- CUDBG\_ERROR\_COMMUNICATION\_FAILURE* Communication error between the debugger and the application.
- CUDBG\_ERROR\_INVALID\_CONTEXT* Specified context cannot be found.
- CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM* Requested address was not originally allocated from device memory (most likely visible in system memory).
- CUDBG\_ERROR\_OS\_RESOURCES* Error while allocating resources from the OS.
- CUDBG\_ERROR\_FORK\_FAILED* Error while forking the debugger process.

### 7.1.3 Function Documentation

#### 7.1.3.1 CUDBGResult cudbgGetAPIVersion (uint32\_t \* *major*, uint32\_t \* *minor*, uint32\_t \* *rev*)

Get the API version supported by the CUDA driver.

**Parameters:**

*major* - the major version number

*minor* - the minor version number

*rev* - the revision version number

**Returns:**

CUDBG\_ERROR\_INVALID\_ARGS,  
CUDBG\_SUCCESS

**See also:**

cudbgGetAPI

# Index

- acknowledgeEvent30
  - EVENT, 58
- acknowledgeEvents42
  - EVENT, 58
- acknowledgeSyncEvents
  - EVENT, 58
- BP
  - setBreakpoint, 16
  - setBreakpoint31, 16
  - unsetBreakpoint, 17
  - unsetBreakpoint31, 17
- Breakpoints, 16
- clearAttachState
  - CUDBGAPI\_st, 67
- cudaDebugger.h, 103
  - CUDBG\_ATTR\_GRID\_LAUNCH\_BLOCKING, 107
  - CUDBG\_ATTR\_GRID\_TID, 107
  - CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_MEM, 109
  - CUDBG\_ERROR\_ALL\_DEVICES\_WATCHDOGGED, 109
  - CUDBG\_ERROR\_BUFFER\_TOO\_SMALL, 109
  - CUDBG\_ERROR\_COMMUNICATION\_FAILURE, 109
  - CUDBG\_ERROR\_FORK\_FAILED, 109
  - CUDBG\_ERROR\_INCOMPATIBLE\_API, 109
  - CUDBG\_ERROR\_INITIALIZATION\_FAILURE, 109
  - CUDBG\_ERROR\_INTERNAL, 109
  - CUDBG\_ERROR\_INVALID\_ADDRESS, 109
  - CUDBG\_ERROR\_INVALID\_ARGS, 109
  - CUDBG\_ERROR\_INVALID\_ATTRIBUTE, 109
  - CUDBG\_ERROR\_INVALID\_CALL\_LEVEL, 109
  - CUDBG\_ERROR\_INVALID\_CONTEXT, 109
  - CUDBG\_ERROR\_INVALID\_COORDINATES, 109
  - CUDBG\_ERROR\_INVALID\_DEVICE, 109
  - CUDBG\_ERROR\_INVALID\_GRID, 109
  - CUDBG\_ERROR\_INVALID\_LANE, 109
  - CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS, 109
  - CUDBG\_ERROR\_INVALID\_MEMORY\_SEGMENT, 109
  - CUDBG\_ERROR\_INVALID\_SM, 109
  - CUDBG\_ERROR\_INVALID\_WARP, 109
  - CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED, 109
  - CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE, 109
  - CUDBG\_ERROR\_OS\_RESOURCES, 109
  - CUDBG\_ERROR\_RUNNING\_DEVICE, 109
  - CUDBG\_ERROR\_SOME\_DEVICES\_WATCHDOGGED, 109
  - CUDBG\_ERROR\_SUSPENDED\_DEVICE, 109
  - CUDBG\_ERROR\_UNINITIALIZED, 109
  - CUDBG\_ERROR\_UNKNOWN, 109
  - CUDBG\_ERROR\_UNKNOWN\_FUNCTION, 109
  - CUDBG\_ERROR\_ZERO\_CALL\_DEPTH, 109
  - CUDBG\_EXCEPTION\_DEVICE\_HARDWARE\_STACK\_OVERFLOW, 107
  - CUDBG\_EXCEPTION\_DEVICE\_ILLEGAL\_ADDRESS, 107
  - CUDBG\_EXCEPTION\_LANE\_ILLEGAL\_ADDRESS, 107
  - CUDBG\_EXCEPTION\_LANE\_MISALIGNED\_ADDRESS, 108
  - CUDBG\_EXCEPTION\_LANE\_USER\_STACK\_OVERFLOW, 107
  - CUDBG\_EXCEPTION\_NONE, 107
  - CUDBG\_EXCEPTION\_UNKNOWN, 107
  - CUDBG\_EXCEPTION\_WARP\_HARDWARE\_STACK\_OVERFLOW, 107
  - CUDBG\_EXCEPTION\_WARP\_ILLEGAL\_INSTRUCTION, 107
  - CUDBG\_EXCEPTION\_WARP\_INVALID\_ADDRESS\_SPACE, 107
  - CUDBG\_EXCEPTION\_WARP\_INVALID\_PC, 107
  - CUDBG\_EXCEPTION\_WARP\_MISALIGNED\_ADDRESS, 107
  - CUDBG\_EXCEPTION\_WARP\_OUT\_OF\_RANGE\_ADDRESS, 107
  - CUDBG\_GRID\_STATUS\_ACTIVE, 108
  - CUDBG\_GRID\_STATUS\_INVALID, 108
  - CUDBG\_GRID\_STATUS\_PENDING, 108
  - CUDBG\_GRID\_STATUS\_SLEEPING, 108
  - CUDBG\_GRID\_STATUS\_TERMINATED, 108
  - CUDBG\_GRID\_STATUS\_UNDETERMINED, 108
  - CUDBG\_KNL\_TYPE\_APPLICATION, 108

- CUDBG\_KNL\_TYPE\_SYSTEM, 108
- CUDBG\_KNL\_TYPE\_UNKNOWN, 108
- CUDBG\_SUCCESS, 109
- CUDBGAttribute, 107
- CUDBGException\_t, 107
- cudaGetAPIVersion, 110
- CUDBGGridStatus, 108
- CUDBGKernelType, 108
- CUDBGRegClass, 108
- CUDBGResult, 108
- REG\_CLASS\_INVALID, 108
- REG\_CLASS\_LMEM\_REG\_OFFSET, 108
- REG\_CLASS\_MEM\_LOCAL, 108
- REG\_CLASS\_REG\_ADDR, 108
- REG\_CLASS\_REG\_CC, 108
- REG\_CLASS\_REG\_FULL, 108
- REG\_CLASS\_REG\_HALF, 108
- REG\_CLASS\_REG\_PRED, 108
- CUDBG\_ATTR\_GRID\_LAUNCH\_BLOCKING  
  cudaDebugger.h, 107
- CUDBG\_ATTR\_GRID\_TID  
  cudaDebugger.h, 107
- CUDBG\_ERROR\_ADDRESS\_NOT\_IN\_DEVICE\_-  
  MEM  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_ALL\_DEVICES\_WATCHDOGGED  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_BUFFER\_TOO\_SMALL  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_COMMUNICATION\_FAILURE  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_FORK\_FAILED  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INCOMPATIBLE\_API  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INITIALIZATION\_FAILURE  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INTERNAL  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_ADDRESS  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_ARGS  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_ATTRIBUTE  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_CALL\_LEVEL  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_CONTEXT  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_COORDINATES  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_DEVICE  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_GRID  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_LANE  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_MEMORY\_ACCESS  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_MEMORY\_SEGMENT  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_SM  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_INVALID\_WARP  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_MEMORY\_MAPPING\_FAILED  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_NO\_EVENT\_AVAILABLE  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_OS\_RESOURCES  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_RUNNING\_DEVICE  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_SOME\_DEVICES\_-  
  WATCHDOGGED  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_SUSPENDED\_DEVICE  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_UNINITIALIZED  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_UNKNOWN  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_UNKNOWN\_FUNCTION  
  cudaDebugger.h, 109
- CUDBG\_ERROR\_ZERO\_CALL\_DEPTH  
  cudaDebugger.h, 109
- CUDBG\_EVENT\_ATTACH\_COMPLETE  
  EVENT, 58
- CUDBG\_EVENT\_CTX\_CREATE  
  EVENT, 57
- CUDBG\_EVENT\_CTX\_DESTROY  
  EVENT, 57
- CUDBG\_EVENT\_CTX\_POP  
  EVENT, 57
- CUDBG\_EVENT\_CTX\_PUSH  
  EVENT, 57
- CUDBG\_EVENT\_ELF\_IMAGE\_LOADED  
  EVENT, 57
- CUDBG\_EVENT\_INTERNAL\_ERROR  
  EVENT, 57
- CUDBG\_EVENT\_INVALID  
  EVENT, 57
- CUDBG\_EVENT\_KERNEL\_FINISHED  
  EVENT, 57
- CUDBG\_EVENT\_KERNEL\_READY  
  EVENT, 57
- CUDBG\_EVENT\_TIMEOUT  
  EVENT, 57

- CUDBG\_EXCEPTION\_DEVICE\_HARDWARE\_-  
STACK\_OVERFLOW  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_DEVICE\_ILLEGAL\_-  
ADDRESS  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_LANE\_ILLEGAL\_ADDRESS  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_LANE\_MISALIGNED\_-  
ADDRESS  
  [cudadebugger.h, 108](#)
- CUDBG\_EXCEPTION\_LANE\_USER\_STACK\_-  
OVERFLOW  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_NONE  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_UNKNOWN  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_WARP\_HARDWARE\_-  
STACK\_OVERFLOW  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_WARP\_ILLEGAL\_-  
INSTRUCTION  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_WARP\_INVALID\_-  
ADDRESS\_SPACE  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_WARP\_INVALID\_PC  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_WARP\_MISALIGNED\_-  
ADDRESS  
  [cudadebugger.h, 107](#)
- CUDBG\_EXCEPTION\_WARP\_OUT\_OF\_RANGE\_-  
ADDRESS  
  [cudadebugger.h, 107](#)
- CUDBG\_GRID\_STATUS\_ACTIVE  
  [cudadebugger.h, 108](#)
- CUDBG\_GRID\_STATUS\_INVALID  
  [cudadebugger.h, 108](#)
- CUDBG\_GRID\_STATUS\_PENDING  
  [cudadebugger.h, 108](#)
- CUDBG\_GRID\_STATUS\_SLEEPING  
  [cudadebugger.h, 108](#)
- CUDBG\_GRID\_STATUS\_TERMINATED  
  [cudadebugger.h, 108](#)
- CUDBG\_GRID\_STATUS\_UNDETERMINED  
  [cudadebugger.h, 108](#)
- CUDBG\_KNL\_TYPE\_APPLICATION  
  [cudadebugger.h, 108](#)
- CUDBG\_KNL\_TYPE\_SYSTEM  
  [cudadebugger.h, 108](#)
- CUDBG\_KNL\_TYPE\_UNKNOWN  
  [cudadebugger.h, 108](#)
- CUDBG\_SUCCESS  
  [cudadebugger.h, 109](#)
- CUDBGAPI\_st, [61](#)
- [clearAttachState, 67](#)
- [getGridStatus, 67](#)
- [requestCleanupOnDetach, 67](#)
- CUDBGAttribute  
  [cudadebugger.h, 107](#)
- CUDBGEvent, [68](#)
- CUDBGEvent30, [69](#)
- CUDBGEvent30::cases30\_st, [70](#)
- CUDBGEvent30::cases30\_st::elfImageLoaded30\_st, [71](#)
- CUDBGEvent30::cases30\_st::kernelFinished30\_st, [72](#)
- CUDBGEvent30::cases30\_st::kernelReady30\_st, [73](#)
- CUDBGEvent32, [74](#)
- CUDBGEvent32::cases32\_st, [75](#)
- CUDBGEvent32::cases32\_st::contextCreate32\_st, [76](#)
- CUDBGEvent32::cases32\_st::contextDestroy32\_st, [77](#)
- CUDBGEvent32::cases32\_st::contextPop32\_st, [78](#)
- CUDBGEvent32::cases32\_st::contextPush32\_st, [79](#)
- CUDBGEvent32::cases32\_st::elfImageLoaded32\_st, [80](#)
- CUDBGEvent32::cases32\_st::kernelFinished32\_st, [81](#)
- CUDBGEvent32::cases32\_st::kernelReady32\_st, [82](#)
- CUDBGEvent42, [83](#)
- CUDBGEvent42::cases42\_st, [84](#)
- CUDBGEvent42::cases42\_st::contextCreate42\_st, [85](#)
- CUDBGEvent42::cases42\_st::contextDestroy42\_st, [86](#)
- CUDBGEvent42::cases42\_st::contextPop42\_st, [87](#)
- CUDBGEvent42::cases42\_st::contextPush42\_st, [88](#)
- CUDBGEvent42::cases42\_st::elfImageLoaded42\_st, [89](#)
- CUDBGEvent42::cases42\_st::kernelFinished42\_st, [90](#)
- CUDBGEvent42::cases42\_st::kernelReady42\_st, [91](#)
- CUDBGEvent::cases\_st, [92](#)
- CUDBGEvent::cases\_st::contextCreate\_st, [93](#)
- CUDBGEvent::cases\_st::contextDestroy\_st, [94](#)
- CUDBGEvent::cases\_st::contextPop\_st, [95](#)
- CUDBGEvent::cases\_st::contextPush\_st, [96](#)
- CUDBGEvent::cases\_st::elfImageLoaded\_st, [97](#)
- CUDBGEvent::cases\_st::internalError\_st, [98](#)
- CUDBGEvent::cases\_st::kernelFinished\_st, [99](#)
- CUDBGEvent::cases\_st::kernelReady\_st, [100](#)
- CUDBGEventCallbackData, [101](#)
- CUDBGEventCallbackData40, [102](#)
- CUDBGEventKind  
  EVENT, [57](#)
- CUDBGException\_t  
  [cudadebugger.h, 107](#)
- cudbgGetAPIVersion  
  [cudadebugger.h, 110](#)
- CUDBGGridStatus  
  [cudadebugger.h, 108](#)
- CUDBGKernelType  
  [cudadebugger.h, 108](#)
- CUDBGRegClass  
  [cudadebugger.h, 108](#)

- CUDBGResult
  - cudaDebugger.h, 108
- DEV
  - getDeviceType, 48
  - getNumDevices, 48
  - getNumLanes, 49
  - getNumRegisters, 49
  - getNumSMs, 50
  - getNumWarps, 50
  - getSmType, 51
- Device Execution Control, 13
- Device Properties, 48
- Device State Alteration, 39
- Device State Inspection, 18
- disassemble
  - DWARF, 52
- DWARF
  - disassemble, 52
  - getHostAddrFromDeviceAddr, 52
  - getPhysicalRegister30, 53
  - getPhysicalRegister40, 53
  - isDeviceCodeAddress, 54
  - lookupDeviceCodeSymbol, 54
- DWARF Utilities, 52
- EVENT
  - acknowledgeEvent30, 58
  - acknowledgeEvents42, 58
  - acknowledgeSyncEvents, 58
  - CUDBG\_EVENT\_ATTACH\_COMPLETE, 58
  - CUDBG\_EVENT\_CTX\_CREATE, 57
  - CUDBG\_EVENT\_CTX\_DESTROY, 57
  - CUDBG\_EVENT\_CTX\_POP, 57
  - CUDBG\_EVENT\_CTX\_PUSH, 57
  - CUDBG\_EVENT\_ELF\_IMAGE\_LOADED, 57
  - CUDBG\_EVENT\_INTERNAL\_ERROR, 57
  - CUDBG\_EVENT\_INVALID, 57
  - CUDBG\_EVENT\_KERNEL\_FINISHED, 57
  - CUDBG\_EVENT\_KERNEL\_READY, 57
  - CUDBG\_EVENT\_TIMEOUT, 57
  - CUDBGEventKind, 57
  - getNextAsyncEvent, 58
  - getNextEvent30, 58
  - getNextEvent32, 59
  - getNextEvent42, 59
  - getNextSyncEvent, 59
  - setNotifyNewEventCallback, 59
  - setNotifyNewEventCallback31, 60
  - setNotifyNewEventCallback40, 60
- Events, 55
- EXEC
  - resumeDevice, 13
  - singleStepWarp, 13
- singleStepWarp40, 14
- suspendDevice, 14
- finalize
  - INIT, 11
- getBlockDim
  - GRID, 44
- getDeviceType
  - DEV, 48
- getElfImage
  - GRID, 45
- getElfImage32
  - GRID, 45
- getGridAttribute
  - GRID, 45
- getGridAttributes
  - GRID, 46
- getGridDim
  - GRID, 46
- getGridDim32
  - GRID, 47
- getGridStatus
  - CUDBGAPI\_st, 67
- getHostAddrFromDeviceAddr
  - DWARF, 52
- getNextAsyncEvent
  - EVENT, 58
- getNextEvent30
  - EVENT, 58
- getNextEvent32
  - EVENT, 59
- getNextEvent42
  - EVENT, 59
- getNextSyncEvent
  - EVENT, 59
- getNumDevices
  - DEV, 48
- getNumLanes
  - DEV, 49
- getNumRegisters
  - DEV, 49
- getNumSMs
  - DEV, 50
- getNumWarps
  - DEV, 50
- getPhysicalRegister30
  - DWARF, 53
- getPhysicalRegister40
  - DWARF, 53
- getSmType
  - DEV, 51
- getTID
  - GRID, 47

## GRID

- getBlockDim, [44](#)
- getElfImage, [45](#)
- getElfImage32, [45](#)
- getGridAttribute, [45](#)
- getGridAttributes, [46](#)
- getGridDim, [46](#)
- getGridDim32, [47](#)
- getTID, [47](#)

- Grid Properties, [44](#)

## INIT

- finalize, [11](#)
- initialize, [11](#)

- Initialization, [11](#)

- initialize

- INIT, [11](#)

- isDeviceCodeAddress

- DWARF, [54](#)

- lookupDeviceCodeSymbol

- DWARF, [54](#)

- memcheckReadErrorAddress

- READ, [20](#)

## READ

- memcheckReadErrorAddress, [20](#)

- readActiveLanes, [20](#)

- readBlockIdx, [21](#)

- readBlockIdx32, [22](#)

- readBrokenWarps, [22](#)

- readCallDepth, [23](#)

- readCallDepth32, [23](#)

- readCodeMemory, [24](#)

- readConstMemory, [24](#)

- readGlobalMemory, [25](#)

- readGlobalMemory31, [25](#)

- readGridId, [26](#)

- readLaneException, [27](#)

- readLaneStatus, [27](#)

- readLocalMemory, [27](#)

- readParamMemory, [28](#)

- readPC, [29](#)

- readPinnedMemory, [29](#)

- readRegister, [30](#)

- readReturnAddress, [31](#)

- readReturnAddress32, [31](#)

- readSharedMemory, [32](#)

- readSyscallCallDepth, [32](#)

- readTextureMemory, [33](#)

- readTextureMemoryBindless, [34](#)

- readThreadId, [34](#)

- readValidLanes, [35](#)

- readValidWarps, [36](#)

- readVirtualPC, [36](#)

- readVirtualReturnAddress, [37](#)

- readVirtualReturnAddress32, [37](#)

- writePinnedMemory, [38](#)

- readActiveLanes

- READ, [20](#)

- readBlockIdx

- READ, [21](#)

- readBlockIdx32

- READ, [22](#)

- readBrokenWarps

- READ, [22](#)

- readCallDepth

- READ, [23](#)

- readCallDepth32

- READ, [23](#)

- readCodeMemory

- READ, [24](#)

- readConstMemory

- READ, [24](#)

- readGlobalMemory

- READ, [25](#)

- readGlobalMemory31

- READ, [25](#)

- readGridId

- READ, [26](#)

- readLaneException

- READ, [27](#)

- readLaneStatus

- READ, [27](#)

- readLocalMemory

- READ, [27](#)

- readParamMemory

- READ, [28](#)

- readPC

- READ, [29](#)

- readPinnedMemory

- READ, [29](#)

- readRegister

- READ, [30](#)

- readReturnAddress

- READ, [31](#)

- readReturnAddress32

- READ, [31](#)

- readSharedMemory

- READ, [32](#)

- readSyscallCallDepth

- READ, [32](#)

- readTextureMemory

- READ, [33](#)

- readTextureMemoryBindless

- READ, [34](#)

- readThreadId

- READ, [34](#)

- readValidLanes
  - READ, [35](#)
- readValidWarps
  - READ, [36](#)
- readVirtualPC
  - READ, [36](#)
- readVirtualReturnAddress
  - READ, [37](#)
- readVirtualReturnAddress32
  - READ, [37](#)
- REG\_CLASS\_INVALID
  - cudaDebugger.h, [108](#)
- REG\_CLASS\_LMEM\_REG\_OFFSET
  - cudaDebugger.h, [108](#)
- REG\_CLASS\_MEM\_LOCAL
  - cudaDebugger.h, [108](#)
- REG\_CLASS\_REG\_ADDR
  - cudaDebugger.h, [108](#)
- REG\_CLASS\_REG\_CC
  - cudaDebugger.h, [108](#)
- REG\_CLASS\_REG\_FULL
  - cudaDebugger.h, [108](#)
- REG\_CLASS\_REG\_HALF
  - cudaDebugger.h, [108](#)
- REG\_CLASS\_REG\_PRED
  - cudaDebugger.h, [108](#)
- requestCleanupOnDetach
  - CUDBGAPI\_st, [67](#)
- resumeDevice
  - EXEC, [13](#)
  
- setBreakpoint
  - BP, [16](#)
- setBreakpoint31
  - BP, [16](#)
- setNotifyNewEventCallback
  - EVENT, [59](#)
- setNotifyNewEventCallback31
  - EVENT, [60](#)
- setNotifyNewEventCallback40
  - EVENT, [60](#)
- singleStepWarp
  - EXEC, [13](#)
- singleStepWarp40
  - EXEC, [14](#)
- suspendDevice
  - EXEC, [14](#)
  
- unsetBreakpoint
  - BP, [17](#)
- unsetBreakpoint31
  - BP, [17](#)
  
- WRITE
  - writeGlobalMemory, [39](#)
  - writeGlobalMemory31, [40](#)
  - writeLocalMemory, [40](#)
  - writeParamMemory, [41](#)
  - writeRegister, [41](#)
  - writeSharedMemory, [42](#)