



NVIDIA CUDA DEBUGGER API

Reference Manual

Version 4.2

May 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

Contents

1	Introduction	1
1.1	Debugger API	1
1.2	ELF and DWARF	2
1.3	ABI Support	3
1.4	Exception Reporting	3
2	Module Index	5
2.1	Modules	5
3	Data Structure Index	7
3.1	Data Structures	7
4	File Index	9
4.1	File List	9
5	Module Documentation	11
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	15
5.3.1	Detailed Description	15

5.3.2	Variable Documentation	15
5.3.2.1	setBreakpoint	15
5.3.2.2	setBreakpoint31	15
5.3.2.3	unsetBreakpoint	16
5.3.2.4	unsetBreakpoint31	16
5.4	Device State Inspection	17
5.4.1	Detailed Description	19
5.4.2	Variable Documentation	19
5.4.2.1	readActiveLanes	19
5.4.2.2	readBlockIdx	20
5.4.2.3	readBlockIdx32	20
5.4.2.4	readBrokenWarps	21
5.4.2.5	readCallDepth	21
5.4.2.6	readCallDepth32	22
5.4.2.7	readCodeMemory	22
5.4.2.8	readConstMemory	23
5.4.2.9	readGlobalMemory	23
5.4.2.10	readGlobalMemory31	24
5.4.2.11	readGridId	24
5.4.2.12	readLaneException	25
5.4.2.13	readLaneStatus	25
5.4.2.14	readLocalMemory	26
5.4.2.15	readParamMemory	26
5.4.2.16	readPC	27
5.4.2.17	readPinnedMemory	27
5.4.2.18	readRegister	28
5.4.2.19	readReturnAddress	29
5.4.2.20	readReturnAddress32	29
5.4.2.21	readSharedMemory	30
5.4.2.22	readSyscallCallDepth	30
5.4.2.23	readTextureMemory	31
5.4.2.24	readTextureMemoryBindless	31
5.4.2.25	readThreadId	32
5.4.2.26	readValidLanes	33
5.4.2.27	readValidWarps	33
5.4.2.28	readVirtualPC	34
5.4.2.29	readVirtualReturnAddress	34

5.4.2.30	readVirtualReturnAddress32	35
5.4.2.31	writePinnedMemory	35
5.5	Device State Alteration	37
5.5.1	Detailed Description	37
5.5.2	Variable Documentation	37
5.5.2.1	writeGlobalMemory	37
5.5.2.2	writeGlobalMemory31	38
5.5.2.3	writeLocalMemory	38
5.5.2.4	writeParamMemory	39
5.5.2.5	writeRegister	39
5.5.2.6	writeSharedMemory	40
5.6	Grid Properties	41
5.6.1	Detailed Description	41
5.6.2	Variable Documentation	41
5.6.2.1	getBlockDim	41
5.6.2.2	getElfImage	42
5.6.2.3	getElfImage32	42
5.6.2.4	getGridAttribute	42
5.6.2.5	getGridAttributes	43
5.6.2.6	getGridDim	43
5.6.2.7	getGridDim32	44
5.6.2.8	getTID	44
5.7	Device Properties	45
5.7.1	Detailed Description	45
5.7.2	Variable Documentation	45
5.7.2.1	getDeviceType	45
5.7.2.2	getNumDevices	46
5.7.2.3	getNumLanes	46
5.7.2.4	getNumRegisters	46
5.7.2.5	getNumSMs	47
5.7.2.6	getNumWarps	47
5.7.2.7	getSmType	47
5.8	DWARF Utilities	49
5.8.1	Detailed Description	49
5.8.2	Variable Documentation	49
5.8.2.1	disassemble	49
5.8.2.2	getHostAddrFromDeviceAddr	50

5.8.2.3	getPhysicalRegister30	50
5.8.2.4	getPhysicalRegister40	50
5.8.2.5	isDeviceCodeAddress	51
5.8.2.6	lookupDeviceCodeSymbol	51
5.9	Events	52
5.9.1	Detailed Description	53
5.9.2	Enumeration Type Documentation	54
5.9.2.1	CUDBGEventKind	54
5.9.3	Variable Documentation	54
5.9.3.1	acknowledgeEvent30	54
5.9.3.2	acknowledgeEvents	54
5.9.3.3	getNextEvent	54
5.9.3.4	getNextEvent30	55
5.9.3.5	getNextEvent32	55
5.9.3.6	setNotifyNewEventCallback	55
5.9.3.7	setNotifyNewEventCallback31	55
5.9.3.8	setNotifyNewEventCallback40	55
6	Data Structure Documentation	57
6.1	CUDBGAPI_st Struct Reference	57
6.2	CUDBGEvent Struct Reference	63
6.2.1	Field Documentation	64
6.2.1.1	context	64
6.2.1.2	dev	65
6.2.1.3	tid	65
6.3	CUDBGEvent30 Struct Reference	66
6.4	CUDBGEventCallbackData Struct Reference	67
7	File Documentation	69
7.1	cuda debugger.h File Reference	69
7.1.1	Detailed Description	72
7.1.2	Enumeration Type Documentation	72
7.1.2.1	CUDBGAttribute	72
7.1.2.2	CUDBGException_t	72
7.1.2.3	CUDBGKernelType	72
7.1.2.4	CUDBGRegClass	73
7.1.2.5	CUDBGResult	73
7.1.3	Function Documentation	74

7.1.3.1	cudbgGetAPI	74
7.1.3.2	cudbgGetAPIVersion	74

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	15
Device State Inspection	17
Device State Alteration	37
Grid Properties	41
Device Properties	45
DWARF Utilities	49
Events	52

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

CUDBGAPI_st (The CUDA debugger API routines)	57
CUDBGEvent (Event information container)	63
CUDBGEvent30 (Event information container (deprecated, 3.0 only))	66
CUDBGEventCallbackData (Event information passed to callback set with setNotifyNewEventCallback function)	67

Chapter 4

File Index

4.1 File List

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

[cudadebugger.h](#) (Header file for the CUDA debugger API) 69

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 CUDBGAPI_st::finalize [inherited]

Returns:

CUDBG_SUCCESS
CUDBG_ERROR_UNINITIALIZED
CUDBG_ERROR_COMMUNICATION_FAILURE
CUDBG_ERROR_UNKNOWN

See also:

[initialize](#)

5.1.2.2 CUDBGAPI_st::initialize [inherited]

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 CUDBGAPI_st::resumeDevice [inherited]

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 CUDBGAPI_st::singleStepWarp [inherited]

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 CUDBGAPI_st::singleStepWarp40 [inherited]**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 CUDBGAPI_st::suspendDevice [inherited]**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 CUDBGAPI_st::setBreakpoint [inherited]

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 CUDBGAPI_st::setBreakpoint31 [inherited]

Parameters:

addr - instruction address

Returns:

CUDBG_SUCCESS
CUDBG_ERROR_UNINITIALIZED
CUDBG_ERROR_INVALID_ADDRESS

See also:

[unsetBreakpoint31](#)

5.3.2.3 CUDBGAPI_st::unsetBreakpoint [inherited]

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 CUDBGAPI_st::unsetBreakpoint31 [inherited]

Parameters:

addr - instruction address

Returns:

CUDBG_SUCCESS
CUDBG_ERROR_UNINITIALIZED

See also:

[setBreakpoint31](#)

5.4 Device State Inspection

Variables

- `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 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(* CUDBGAPI_st::readThreadIdx)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CuDim3 *threadIdx)`
Reads the CUDA thread index running on valid lane.
- `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 CUDBGAPI_st::readActiveLanes [inherited]

Parameters:

dev - device index

sm - SM index

wp - warp index

activeLanes - 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](#)

5.4.2.2 CUDBGAPI_st::readBlockIdx [inherited]

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](#)

5.4.2.3 CUDBGAPI_st::readBlockIdx32 [inherited]

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](#)

5.4.2.4 CUDBGAPI_st::readBrokenWarps [inherited]

Parameters:

dev - device index

sm - SM index

brokenWarps - 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](#)

5.4.2.5 CUDBGAPI_st::readCallDepth [inherited]

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](#)

5.4.2.6 CUDBGAPI_st::readCallDepth32 [inherited]

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](#)

5.4.2.7 CUDBGAPI_st::readCodeMemory [inherited]

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](#)

5.4.2.8 CUDBGAPI_st::readConstMemory [inherited]

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](#)

5.4.2.9 CUDBGAPI_st::readGlobalMemory [inherited]

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](#)

5.4.2.10 CUDBGAPI_st::readGlobalMemory31 [inherited]**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](#)

5.4.2.11 CUDBGAPI_st::readGridId [inherited]**Parameters:**

dev - device index
sm - SM index
wp - warp index
threadIdx - 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](#)

5.4.2.12 CUDBGAPI_st::readLaneException [inherited]

Parameters:

dev - device index
sm - SM index
wp - warp index
ln - lane index
error - 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

5.4.2.13 CUDBGAPI_st::readLaneStatus [inherited]

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

5.4.2.14 CUDBGAPI_st::readLocalMemory [inherited]

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](#)

5.4.2.15 CUDBGAPI_st::readParamMemory [inherited]

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](#)

5.4.2.16 CUDBGAPI_st::readPC [inherited]**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](#)

5.4.2.17 CUDBGAPI_st::readPinnedMemory [inherited]**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](#)

5.4.2.18 CUDBGAPI_st::readRegister [inherited]**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](#)

5.4.2.19 CUDBGAPI_st::readReturnAddress [inherited]

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](#)

5.4.2.20 CUDBGAPI_st::readReturnAddress32 [inherited]

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](#)

5.4.2.21 CUDBGAPI_st::readSharedMemory [inherited]

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](#)

5.4.2.22 CUDBGAPI_st::readSyscallCallDepth [inherited]

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](#)

5.4.2.23 CUDBGAPI_st::readTextureMemory [inherited]

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:

dev - device index
sm - 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](#)

5.4.2.24 CUDBGAPI_st::readTextureMemoryBindless [inherited]

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:

dev - device index
sm - 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](#)

5.4.2.25 CUDBGAPI_st::readThreadIdx [inherited]**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](#)

5.4.2.26 CUDBGAPI_st::readValidLanes [inherited]

Parameters:

dev - device index
sm - SM index
wp - warp index
validLanes - 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](#)

5.4.2.27 CUDBGAPI_st::readValidWarps [inherited]

Parameters:

dev - device index
sm - SM index
validWarps - 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](#)

5.4.2.28 CUDBGAPI_st::readVirtualPC [inherited]

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](#)

5.4.2.29 CUDBGAPI_st::readVirtualReturnAddress [inherited]

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](#)

5.4.2.30 CUDBGAPI_st::readVirtualReturnAddress32 [inherited]

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](#)

5.4.2.31 CUDBGAPI_st::writePinnedMemory [inherited]

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 CUDBGAPI_st::writeGlobalMemory [inherited]

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 CUDBGAPI_st::writeGlobalMemory31 [inherited]

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 CUDBGAPI_st::writeLocalMemory [inherited]

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 CUDBGAPI_st::writeParamMemory [inherited]**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 CUDBGAPI_st::writeRegister [inherited]**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 CUDBGAPI_st::writeSharedMemory [inherited]**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 CUDBGAPI_st::getBlockDim [inherited]

Parameters:

devId - 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 CUDBGAPI_st::getElfImage [inherited]

Parameters:

devId - 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 CUDBGAPI_st::getElfImage32 [inherited]

Parameters:

devId - 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 CUDBGAPI_st::getGridAttribute [inherited]

Parameters:

devId - 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 CUDBGAPI_st::getGridAttributes [inherited]**Parameters:**

devId - 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 CUDBGAPI_st::getGridDim [inherited]**Parameters:**

devId - 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 CUDBGAPI_st::getGridDim32 [inherited]

Parameters:

devId - 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 CUDBGAPI_st::getTID [inherited]

Parameters:

devId - 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 CUDBGAPI_st::getDeviceType [inherited]

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 CUDBGAPI_st::getNumDevices [inherited]

Parameters:

numDevs - 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 CUDBGAPI_st::getNumLanes [inherited]

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 CUDBGAPI_st::getNumRegisters [inherited]

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 CUDBGAPI_st::getNumSMs [inherited]

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 CUDBGAPI_st::getNumWarps [inherited]

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 CUDBGAPI_st::getSmType [inherited]

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 CUDBGAPI_st::disassemble [inherited]

Parameters:

dev - device index
addr - instruction address
instSize - instruction size (32 or 64 bits)
dis - disassembled instruction buffer
disSize - disassembled instruction buffer size

Returns:

CUDBG_SUCCESS
 CUDBG_ERROR_INVALID_ARGS
 CUDBG_ERROR_INVALID_DEVICE
 CUDBG_ERROR_UNKNOWN

5.8.2.2 CUDBGAPI_st::getHostAddrFromDeviceAddr [inherited]

Parameters:

devId - device index
dva - device memory address
hva - 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 CUDBGAPI_st::getPhysicalRegister30 [inherited]

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 CUDBGAPI_st::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. 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 CUDBGAPI_st::isDeviceCodeAddress [inherited]**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 CUDBGAPI_st::lookupDeviceCodeSymbol [inherited]**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, 3.0 only).
- struct [CUDBGEventCallbackData](#)
Event information passed to callback set with `setNotifyNewEventCallback` function.

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](#) }
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::acknowledgeEvents](#))(void)
Inform the debugger API that the events have been processed.
- [CUDBGResult](#)(* [CUDBGAPI_st::getNextEvent](#))(CUDBGEvent *event)
Copies the next available event in the event queue into 'event' and removes it from the queue.
- [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::setNotifyNewEventCallback)(CUDBGNotifyNewEventCallback call-back)`
Provides the API with the function to call to notify the debugger of a new application or device event.
- `CUDBGResult(* CUDBGAPI_st::setNotifyNewEventCallback31)(CUDBGNotifyNewEventCallback31 call-back, 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 call-back)`
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

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.

5.9.3 Variable Documentation

5.9.3.1 CUDBGAPI_st::acknowledgeEvent30 [inherited]

Parameters:

event - pointer to the event that has been processed

Returns:

CUDBG_SUCCESS

5.9.3.2 CUDBGAPI_st::acknowledgeEvents [inherited]

Returns:

CUDBG_SUCCESS

5.9.3.3 CUDBGAPI_st::getNextEvent [inherited]

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

5.9.3.4 CUDBGAPI_st::getNextEvent30 [inherited]

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 CUDBGAPI_st::getNextEvent32 [inherited]

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 CUDBGAPI_st::setNotifyNewEventCallback [inherited]

Parameters:

callback - the callback function

Returns:

CUDBG_SUCCESS

5.9.3.7 CUDBGAPI_st::setNotifyNewEventCallback31 [inherited]

Parameters:

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

Returns:

CUDBG_SUCCESS

5.9.3.8 CUDBGAPI_st::setNotifyNewEventCallback40 [inherited]

Parameters:

callback - the callback function

Returns:

CUDBG_SUCCESS

Chapter 6

Data Structure Documentation

6.1 CUDBGAPI_st 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\(* acknowledgeEvents\)](#)(void)
Inform the debugger API that the events have been processed.
- [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(* 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(* getNextEvent)(CUDBGEvent *event)`

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

- `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(* 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**(* [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(* 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(* 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\(* resumeDevice\)](#)(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.

6.2 CUDBGEvent Struct Reference

Event information container.

Data Fields

- union {
} [cases](#)

Information for each type of event.

- [CUDBGEventKind](#) [kind](#)

Event type.

- [CuDim3](#) [blockDim](#)

block dimensions of the kernel.

- [uint64_t](#) [context](#)

context of the kernel.

- struct {
} [contextCreate](#)

Information about the context being created.

- struct {
} [contextDestroy](#)

Information about the context being destroyed.

- struct {
} [contextPop](#)

Information about the context being popped.

- struct {
} [contextPush](#)

Information about the context being pushed.

- [uint32_t](#) [dev](#)

device index of the kernel.

- struct {
} [elfImageLoaded](#)

Information about the loaded ELF image.

- [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.
- `struct {`
 `}` [kernelFinished](#)

Information about the kernel that just terminated.
- `struct {`
 `}` [kernelReady](#)

Information about the kernel ready to be launched.
- `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.
- `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.

6.2.1 Field Documentation

6.2.1.1 cases `contextDestroy` `CUDBGEvent::context`

the context being destroyed.

the context being created.

the context being popped.

the context being pushed.

6.2.1.2 cases contextDestroy CUDBGEvent::dev

device index of the context.

6.2.1.3 cases contextDestroy CUDBGEvent::tid

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

6.3 CUDBGEvent30 Struct Reference

Event information container (deprecated, 3.0 only).

Data Fields

- union {
} [cases](#)

Information for each type of event.

- [CUDBGEventKind](#) [kind](#)

Event type.

- [uint32_t](#) [dev](#)

device index of the kernel.

- struct {
} [elfImageLoaded](#)

Information about the loaded ELF image.

- [uint32_t](#) [gridId](#)

grid index of the kernel.

- struct {
} [kernelFinished](#)

Information about the kernel that just terminated.

- struct {
} [kernelReady](#)

Information about the kernel ready to be launched.

- [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.

- [uint32_t](#) [tid](#)

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

6.4 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.

Chapter 7

File Documentation

7.1 cudadebugger.h File Reference

Header file for the CUDA debugger API.

Data Structures

- struct [CUDBGAPI_st](#)
The CUDA debugger API routines.
- struct [CUDBGEvent](#)
Event information container.
- struct [CUDBGEvent30](#)
Event information container (deprecated, 3.0 only).
- struct [CUDBGEventCallbackData](#)
Event information passed to callback set with `setNotifyNewEventCallback` function.

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 }

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 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 }

Result values of all the API routines.

Functions

- **CUDBGResult cudbgGetAPI** (uint32_t major, uint32_t minor, uint32_t rev, **CUDBGAPI** *api)

Get the API associated with the major/minor/revision version numbers.
- **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

7.1.2 Enumeration Type Documentation

7.1.2.1 enum CUDBGAttribute

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

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 CUDBGKernelType

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.

7.1.2.4 enum CUDBGRegClass

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).

7.1.2.5 enum CUDBGResult

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 exclude because they have a watch-dog 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).

7.1.3 Function Documentation

7.1.3.1 CUDBGResult cudbgGetAPI (uint32_t *major*, uint32_t *minor*, uint32_t *rev*, CUDBGAPI * *api*)

Parameters:

major - the major version number

minor - the minor version number

rev - the revision version number

api - the pointer to the API

Returns:

CUDBG_ERROR_INVALID_ARGS

CUDBG_SUCCESS

CUDBG_ERROR_INCOMPATIBLE_API

See also:

[cudbgGetAPIVersion](#)

7.1.3.2 CUDBGResult cudbgGetAPIVersion (uint32_t * *major*, uint32_t * *minor*, uint32_t * *rev*)

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, [54](#)
- acknowledgeEvents
 - EVENT, [54](#)
- BP
 - setBreakpoint, [15](#)
 - setBreakpoint31, [15](#)
 - unsetBreakpoint, [15](#)
 - unsetBreakpoint31, [16](#)
- Breakpoints, [15](#)
- context
 - CUDBGEvent, [64](#)
- cudaDebugger.h, [69](#)
 - CUDBG_ATTR_GRID_LAUNCH_BLOCKING, [72](#)
 - CUDBG_ATTR_GRID_TID, [72](#)
 - CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, [74](#)
 - CUDBG_ERROR_ALL_DEVICES_WATCHDOGGED, [73](#)
 - CUDBG_ERROR_BUFFER_TOO_SMALL, [73](#)
 - CUDBG_ERROR_COMMUNICATION_FAILURE, [74](#)
 - CUDBG_ERROR_INCOMPATIBLE_API, [73](#)
 - CUDBG_ERROR_INITIALIZATION_FAILURE, [73](#)
 - CUDBG_ERROR_INTERNAL, [73](#)
 - CUDBG_ERROR_INVALID_ADDRESS, [73](#)
 - CUDBG_ERROR_INVALID_ARGS, [73](#)
 - CUDBG_ERROR_INVALID_ATTRIBUTE, [74](#)
 - CUDBG_ERROR_INVALID_CALL_LEVEL, [74](#)
 - CUDBG_ERROR_INVALID_CONTEXT, [74](#)
 - CUDBG_ERROR_INVALID_COORDINATES, [73](#)
 - CUDBG_ERROR_INVALID_DEVICE, [73](#)
 - CUDBG_ERROR_INVALID_GRID, [73](#)
 - CUDBG_ERROR_INVALID_LANE, [73](#)
 - CUDBG_ERROR_INVALID_MEMORY_ACCESS, [73](#)
 - CUDBG_ERROR_INVALID_MEMORY_SEGMENT, [73](#)
 - CUDBG_ERROR_INVALID_SM, [73](#)
 - CUDBG_ERROR_INVALID_WARP, [73](#)
 - CUDBG_ERROR_MEMORY_MAPPING_FAILED, [73](#)
 - CUDBG_ERROR_NO_EVENT_AVAILABLE, [73](#)
 - CUDBG_ERROR_RUNNING_DEVICE, [73](#)
 - CUDBG_ERROR_SOME_DEVICES_WATCHDOGGED, [73](#)
 - CUDBG_ERROR_SUSPENDED_DEVICE, [73](#)
 - CUDBG_ERROR_UNINITIALIZED, [73](#)
 - CUDBG_ERROR_UNKNOWN, [73](#)
 - CUDBG_ERROR_UNKNOWN_FUNCTION, [73](#)
 - CUDBG_ERROR_ZERO_CALL_DEPTH, [74](#)
 - CUDBG_EXCEPTION_DEVICE_HARDWARE_STACK_OVERFLOW, [72](#)
 - CUDBG_EXCEPTION_DEVICE_ILLEGAL_ADDRESS, [72](#)
 - CUDBG_EXCEPTION_LANE_ILLEGAL_ADDRESS, [72](#)
 - CUDBG_EXCEPTION_LANE_MISALIGNED_ADDRESS, [72](#)
 - CUDBG_EXCEPTION_LANE_USER_STACK_OVERFLOW, [72](#)
 - CUDBG_EXCEPTION_NONE, [72](#)
 - CUDBG_EXCEPTION_UNKNOWN, [72](#)
 - CUDBG_EXCEPTION_WARP_HARDWARE_STACK_OVERFLOW, [72](#)
 - CUDBG_EXCEPTION_WARP_ILLEGAL_INSTRUCTION, [72](#)
 - CUDBG_EXCEPTION_WARP_INVALID_ADDRESS_SPACE, [72](#)
 - CUDBG_EXCEPTION_WARP_INVALID_PC, [72](#)
 - CUDBG_EXCEPTION_WARP_MISALIGNED_ADDRESS, [72](#)
 - CUDBG_EXCEPTION_WARP_OUT_OF_RANGE_ADDRESS, [72](#)
 - CUDBG_KNL_TYPE_APPLICATION, [72](#)
 - CUDBG_KNL_TYPE_SYSTEM, [72](#)
 - CUDBG_KNL_TYPE_UNKNOWN, [72](#)
 - CUDBG_SUCCESS, [73](#)
 - CUDBGAttribute, [72](#)
 - CUDBGException_t, [72](#)
 - cudbgGetAPI, [74](#)
 - cudbgGetAPIVersion, [74](#)
 - CUDBGKernelType, [72](#)
 - CUDBGRegClass, [72](#)
 - CUDBGResult, [73](#)
 - REG_CLASS_INVALID, [73](#)
 - REG_CLASS_LMEM_REG_OFFSET, [73](#)

- REG_CLASS_MEM_LOCAL, [73](#)
- REG_CLASS_REG_ADDR, [73](#)
- REG_CLASS_REG_CC, [73](#)
- REG_CLASS_REG_FULL, [73](#)
- REG_CLASS_REG_HALF, [73](#)
- REG_CLASS_REG_PRED, [73](#)
- CUDBG_ATTR_GRID_LAUNCH_BLOCKING
 [cudaDebugger.h, 72](#)
- CUDBG_ATTR_GRID_TID
 [cudaDebugger.h, 72](#)
- CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_-
 MEM
 [cudaDebugger.h, 74](#)
- CUDBG_ERROR_ALL_DEVICES_WATCHDOGGED
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_BUFFER_TOO_SMALL
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_COMMUNICATION_FAILURE
 [cudaDebugger.h, 74](#)
- CUDBG_ERROR_INCOMPATIBLE_API
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INITIALIZATION_FAILURE
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INTERNAL
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_ADDRESS
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_ARGS
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_ATTRIBUTE
 [cudaDebugger.h, 74](#)
- CUDBG_ERROR_INVALID_CALL_LEVEL
 [cudaDebugger.h, 74](#)
- CUDBG_ERROR_INVALID_CONTEXT
 [cudaDebugger.h, 74](#)
- CUDBG_ERROR_INVALID_COORDINATES
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_DEVICE
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_GRID
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_LANE
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_MEMORY_ACCESS
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_MEMORY_SEGMENT
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_SM
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_INVALID_WARP
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_MEMORY_MAPPING_FAILED
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_NO_EVENT_AVAILABLE
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_RUNNING_DEVICE
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_SOME_DEVICES_-
 WATCHDOGGED
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_SUSPENDED_DEVICE
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_UNINITIALIZED
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_UNKNOWN
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_UNKNOWN_FUNCTION
 [cudaDebugger.h, 73](#)
- CUDBG_ERROR_ZERO_CALL_DEPTH
 [cudaDebugger.h, 74](#)
- CUDBG_EVENT_ELF_IMAGE_LOADED
 EVENT, [54](#)
- CUDBG_EVENT_INVALID
 EVENT, [54](#)
- CUDBG_EVENT_KERNEL_FINISHED
 EVENT, [54](#)
- CUDBG_EVENT_KERNEL_READY
 EVENT, [54](#)
- CUDBG_EXCEPTION_DEVICE_HARDWARE_-
 STACK_OVERFLOW
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_DEVICE_ILLEGAL_-
 ADDRESS
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_LANE_ILLEGAL_ADDRESS
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_LANE_MISALIGNED_-
 ADDRESS
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_LANE_USER_STACK_-
 OVERFLOW
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_NONE
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_UNKNOWN
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_WARP_HARDWARE_-
 STACK_OVERFLOW
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_WARP_ILLEGAL_-
 INSTRUCTION
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_WARP_INVALID_-
 ADDRESS_SPACE
 [cudaDebugger.h, 72](#)
- CUDBG_EXCEPTION_WARP_INVALID_PC
 [cudaDebugger.h, 72](#)

CUDBG_EXCEPTION_WARP_MISALIGNED_-
 ADDRESS
 cudadebugger.h, 72
 CUDBG_EXCEPTION_WARP_OUT_OF_RANGE_-
 ADDRESS
 cudadebugger.h, 72
 CUDBG_KNL_TYPE_APPLICATION
 cudadebugger.h, 72
 CUDBG_KNL_TYPE_SYSTEM
 cudadebugger.h, 72
 CUDBG_KNL_TYPE_UNKNOWN
 cudadebugger.h, 72
 CUDBG_SUCCESS
 cudadebugger.h, 73
 CUDBGAPI_st, 57
 CUDBGAttribute
 cudadebugger.h, 72
 CUDBGEvent, 63
 context, 64
 dev, 64
 tid, 65
 CUDBGEvent30, 66
 CUDBGEventCallbackData, 67
 CUDBGEventKind
 EVENT, 54
 CUDBGException_t
 cudadebugger.h, 72
 cudbgGetAPI
 cudadebugger.h, 74
 cudbgGetAPIVersion
 cudadebugger.h, 74
 CUDBGKernelType
 cudadebugger.h, 72
 CUDBGRegClass
 cudadebugger.h, 72
 CUDBGResult
 cudadebugger.h, 73

 DEV
 getDeviceType, 45
 getNumDevices, 45
 getNumLanes, 46
 getNumRegisters, 46
 getNumSMs, 47
 getNumWarps, 47
 getSmType, 47
 dev
 CUDBGEvent, 64
 Device Execution Control, 13
 Device Properties, 45
 Device State Alteration, 37
 Device State Inspection, 17
 disassemble
 DWARF, 49

DWARF
 disassemble, 49
 getHostAddrFromDeviceAddr, 49
 getPhysicalRegister30, 50
 getPhysicalRegister40, 50
 isDeviceCodeAddress, 51
 lookupDeviceCodeSymbol, 51
 DWARF Utilities, 49

 EVENT
 acknowledgeEvent30, 54
 acknowledgeEvents, 54
 CUDBG_EVENT_ELF_IMAGE_LOADED, 54
 CUDBG_EVENT_INVALID, 54
 CUDBG_EVENT_KERNEL_FINISHED, 54
 CUDBG_EVENT_KERNEL_READY, 54
 CUDBGEventKind, 54
 getNextEvent, 54
 getNextEvent30, 54
 getNextEvent32, 55
 setNotifyNewEventCallback, 55
 setNotifyNewEventCallback31, 55
 setNotifyNewEventCallback40, 55
 Events, 52
 EXEC
 resumeDevice, 13
 singleStepWarp, 13
 singleStepWarp40, 14
 suspendDevice, 14

 finalize
 INIT, 11

 getBlockDim
 GRID, 41
 getDeviceType
 DEV, 45
 getElfImage
 GRID, 42
 getElfImage32
 GRID, 42
 getGridAttribute
 GRID, 42
 getGridAttributes
 GRID, 43
 getGridDim
 GRID, 43
 getGridDim32
 GRID, 43
 getHostAddrFromDeviceAddr
 DWARF, 49
 getNextEvent
 EVENT, 54
 getNextEvent30
 EVENT, 54

- getNextEvent32
 - EVENT, [55](#)
- getNumDevices
 - DEV, [45](#)
- getNumLanes
 - DEV, [46](#)
- getNumRegisters
 - DEV, [46](#)
- getNumSMs
 - DEV, [47](#)
- getNumWarps
 - DEV, [47](#)
- getPhysicalRegister30
 - DWARF, [50](#)
- getPhysicalRegister40
 - DWARF, [50](#)
- getSmType
 - DEV, [47](#)
- getTID
 - GRID, [44](#)
- GRID
 - getBlockDim, [41](#)
 - getElfImage, [42](#)
 - getElfImage32, [42](#)
 - getGridAttribute, [42](#)
 - getGridAttributes, [43](#)
 - getGridDim, [43](#)
 - getGridDim32, [43](#)
 - getTID, [44](#)
- Grid Properties, [41](#)
- INIT
 - finalize, [11](#)
 - initialize, [11](#)
- Initialization, [11](#)
- initialize
 - INIT, [11](#)
- isDeviceCodeAddress
 - DWARF, [51](#)
- lookupDeviceCodeSymbol
 - DWARF, [51](#)
- READ
 - readActiveLanes, [19](#)
 - readBlockIdx, [19](#)
 - readBlockIdx32, [20](#)
 - readBrokenWarps, [20](#)
 - readCallDepth, [21](#)
 - readCallDepth32, [21](#)
 - readCodeMemory, [22](#)
 - readConstMemory, [22](#)
 - readGlobalMemory, [23](#)
 - readGlobalMemory31, [24](#)
 - readGridId, [24](#)
 - readLaneException, [25](#)
 - readLaneStatus, [25](#)
 - readLocalMemory, [25](#)
 - readParamMemory, [26](#)
 - readPC, [27](#)
 - readPinnedMemory, [27](#)
 - readReturnAddress, [28](#)
 - readReturnAddress32, [29](#)
 - readSharedMemory, [30](#)
 - readSyscallCallDepth, [30](#)
 - readTextureMemory, [31](#)
 - readTextureMemoryBindless, [31](#)
 - readThreadId, [32](#)
 - readValidLanes, [33](#)
 - readValidWarps, [33](#)
 - readVirtualPC, [34](#)
 - readVirtualReturnAddress, [34](#)
 - readVirtualReturnAddress32, [35](#)
 - writePinnedMemory, [35](#)
- readActiveLanes
 - READ, [19](#)
- readBlockIdx
 - READ, [19](#)
- readBlockIdx32
 - READ, [20](#)
- readBrokenWarps
 - READ, [20](#)
- readCallDepth
 - READ, [21](#)
- readCallDepth32
 - READ, [21](#)
- readCodeMemory
 - READ, [22](#)
- readConstMemory
 - READ, [22](#)
- readGlobalMemory
 - READ, [23](#)
- readGlobalMemory31
 - READ, [24](#)
- readGridId
 - READ, [24](#)
- readLaneException
 - READ, [25](#)
- readLaneStatus
 - READ, [25](#)
- readLocalMemory
 - READ, [25](#)
- readParamMemory
 - READ, [26](#)
- readPC
 - READ, [27](#)
- readPinnedMemory
 - READ, [27](#)

readRegister
 [READ, 28](#)
readReturnAddress
 [READ, 28](#)
readReturnAddress32
 [READ, 29](#)
readSharedMemory
 [READ, 30](#)
readSyscallCallDepth
 [READ, 30](#)
readTextureMemory
 [READ, 31](#)
readTextureMemoryBindless
 [READ, 31](#)
readThreadId
 [READ, 32](#)
readValidLanes
 [READ, 33](#)
readValidWarps
 [READ, 33](#)
readVirtualPC
 [READ, 34](#)
readVirtualReturnAddress
 [READ, 34](#)
readVirtualReturnAddress32
 [READ, 35](#)
REG_CLASS_INVALID
 [cudadebugger.h, 73](#)
REG_CLASS_LMEM_REG_OFFSET
 [cudadebugger.h, 73](#)
REG_CLASS_MEM_LOCAL
 [cudadebugger.h, 73](#)
REG_CLASS_REG_ADDR
 [cudadebugger.h, 73](#)
REG_CLASS_REG_CC
 [cudadebugger.h, 73](#)
REG_CLASS_REG_FULL
 [cudadebugger.h, 73](#)
REG_CLASS_REG_HALF
 [cudadebugger.h, 73](#)
REG_CLASS_REG_PRED
 [cudadebugger.h, 73](#)
resumeDevice
 [EXEC, 13](#)

setBreakpoint
 [BP, 15](#)
setBreakpoint31
 [BP, 15](#)
setNotifyNewEventCallback
 [EVENT, 55](#)
setNotifyNewEventCallback31
 [EVENT, 55](#)
setNotifyNewEventCallback40
 [EVENT, 55](#)
singleStepWarp
 [EXEC, 13](#)
singleStepWarp40
 [EXEC, 14](#)
suspendDevice
 [EXEC, 14](#)

tid
 [CUDBGEvent, 65](#)

unsetBreakpoint
 [BP, 15](#)
unsetBreakpoint31
 [BP, 16](#)

WRITE
 [writeGlobalMemory, 37](#)
 [writeGlobalMemory31, 38](#)
 [writeLocalMemory, 38](#)
 [writeParamMemory, 39](#)
 [writeRegister, 39](#)
 [writeSharedMemory, 40](#)
writeGlobalMemory
 [WRITE, 37](#)
writeGlobalMemory31
 [WRITE, 38](#)
writeLocalMemory
 [WRITE, 38](#)
writeParamMemory
 [WRITE, 39](#)
writePinnedMemory
 [READ, 35](#)
writeRegister
 [WRITE, 39](#)
writeSharedMemory
 [WRITE, 40](#)