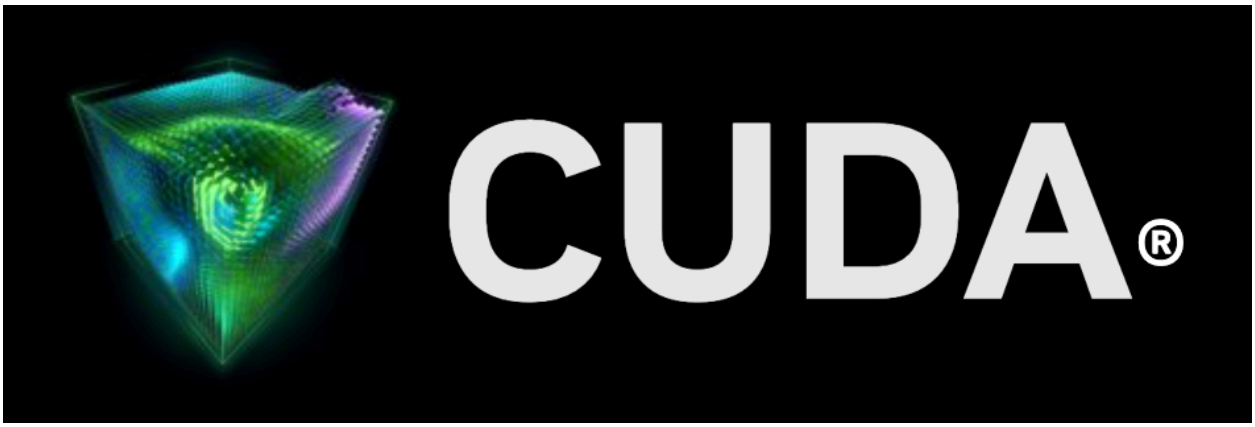


CUDA Debugger API Reference Manual

Release 13.5

NVIDIA Corporation

Nov 02, 2026

Contents

1 Introduction **1**
1.1 Debugger API . . . 1
1.2 ELF and DWARF . . . 2
1.3 ABI Support . . . 3
1.4 Exception Reporting . . . 4
1.5 Attaching and Detaching . . . 4

2 APIs **7**
2.1 Breakpoints . . . 7
2.1.1 Macros . . . 8
2.1.2 Enumerations . . . 9
2.1.3 Typedefs . . . 9
2.2 DWARF Utilities . . . 10
2.2.1 Enumerations . . . 10
2.3 Device Execution Control . . . 12
2.3.1 Enumerations . . . 13
2.4 Device Properties . . . 14
2.5 Device State Alteration . . . 14
2.6 Device State Inspection . . . 15
2.6.1 Enumerations . . . 20
2.7 Events . . . 29
2.7.1 Enumerations . . . 32
2.7.2 Typedefs . . . 34
2.8 General . . . 35
2.8.1 Macros . . . 38
2.8.2 Enumerations . . . 42
2.8.3 Functions . . . 47
2.8.4 Typedefs . . . 49
2.9 Grid Properties . . . 49
2.9.1 Enumerations . . . 50
2.10 Initialization . . . 52

3 Structs **55**
3.1 CUDBGAPI_st . . . 55
3.2 CUDBGAttributeValuePair . . . 197
3.3 CUDBGCbuWarpState . . . 197
3.4 CUDBGCudaLogMessage . . . 198
3.5 CUDBGCudaLogMessage129 . . . 198
3.6 CUDBGCudaLogRule . . . 199
3.7 CUDBGDeviceInfo . . . 200
3.8 CUDBGDeviceInfoSizes . . . 200
3.9 CUDBGEvent . . . 201
3.10 CUDBGEvent30 . . . 208

3.11 CUDBGEvent30::cases30_st::elfImageLoaded30_st . . . 210
3.12 CUDBGEvent30::cases30_st::kernelFinished30_st . . . 211
3.13 CUDBGEvent30::cases30_st::kernelReady30_st . . . 211
3.14 CUDBGEvent32 . . . 211
3.15 CUDBGEvent32::cases32_st::contextCreate32_st . . . 215
3.16 CUDBGEvent32::cases32_st::contextDestroy32_st . . . 215
3.17 CUDBGEvent32::cases32_st::contextPop32_st . . . 216
3.18 CUDBGEvent32::cases32_st::contextPush32_st . . . 216
3.19 CUDBGEvent32::cases32_st::elfImageLoaded32_st . . . 217
3.20 CUDBGEvent32::cases32_st::kernelFinished32_st . . . 217
3.21 CUDBGEvent32::cases32_st::kernelReady32_st . . . 218
3.22 CUDBGEvent42 . . . 218
3.23 CUDBGEvent42::cases42_st::contextCreate42_st . . . 222
3.24 CUDBGEvent42::cases42_st::contextDestroy42_st . . . 222
3.25 CUDBGEvent42::cases42_st::contextPop42_st . . . 223
3.26 CUDBGEvent42::cases42_st::contextPush42_st . . . 223
3.27 CUDBGEvent42::cases42_st::elfImageLoaded42_st . . . 223
3.28 CUDBGEvent42::cases42_st::kernelFinished42_st . . . 224
3.29 CUDBGEvent42::cases42_st::kernelReady42_st . . . 224
3.30 CUDBGEvent50 . . . 225
3.31 CUDBGEvent50::cases50_st::contextCreate50_st . . . 229
3.32 CUDBGEvent50::cases50_st::contextDestroy50_st . . . 230
3.33 CUDBGEvent50::cases50_st::contextPop50_st . . . 230
3.34 CUDBGEvent50::cases50_st::contextPush50_st . . . 230
3.35 CUDBGEvent50::cases50_st::elfImageLoaded50_st . . . 231
3.36 CUDBGEvent50::cases50_st::internalError50_st . . . 231
3.37 CUDBGEvent50::cases50_st::kernelFinished50_st . . . 231
3.38 CUDBGEvent50::cases50_st::kernelReady50_st . . . 232
3.39 CUDBGEvent55 . . . 233
3.40 CUDBGEvent55::cases55_st::contextCreate55_st . . . 237
3.41 CUDBGEvent55::cases55_st::contextDestroy55_st . . . 237
3.42 CUDBGEvent55::cases55_st::contextPop55_st . . . 238
3.43 CUDBGEvent55::cases55_st::contextPush55_st . . . 238
3.44 CUDBGEvent55::cases55_st::elfImageLoaded55_st . . . 239
3.45 CUDBGEvent55::cases55_st::internalError55_st . . . 239
3.46 CUDBGEvent55::cases55_st::kernelFinished55_st . . . 239
3.47 CUDBGEvent55::cases55_st::kernelReady55_st . . . 240
3.48 CUDBGEvent::cases_st::allDevicesSuspended_st . . . 241
3.49 CUDBGEvent::cases_st::contextCreate_st . . . 242
3.50 CUDBGEvent::cases_st::contextDestroy_st . . . 242
3.51 CUDBGEvent::cases_st::contextPop_st . . . 243
3.52 CUDBGEvent::cases_st::contextPush_st . . . 243
3.53 CUDBGEvent::cases_st::cudaLogsRulesetChanged_st . . . 244
3.54 CUDBGEvent::cases_st::elfImageLoaded_st . . . 244
3.55 CUDBGEvent::cases_st::elfImageUnloaded_st . . . 245
3.56 CUDBGEvent::cases_st::functionsLoaded_st . . . 245
3.57 CUDBGEvent::cases_st::internalError_st . . . 246
3.58 CUDBGEvent::cases_st::kernelFinished_st . . . 246
3.59 CUDBGEvent::cases_st::kernelReady_st . . . 247
3.60 CUDBGEvent::cases_st::singleStepComplete_st . . . 248
3.61 CUDBGEventCallbackData . . . 249
3.62 CUDBGEventCallbackData40 . . . 249
3.63 CUDBGEventCallbackData41 . . . 250
3.64 CUDBGGridInfo . . . 250

3.65 CUDBGGridInfo120 . . . 252
3.66 CUDBGGridInfo55 . . . 253
3.67 CUDBGLaneInfo . . . 254
3.68 CUDBGLaneState . . . 254
3.69 CUDBGLoadedFunctionInfo . . . 255
3.70 CUDBGMemoryInfo . . . 255
3.71 CUDBGSMInfo . . . 255
3.72 CUDBGTcgen05TmemInfo . . . 256
3.73 CUDBGWarpInfo . . . 256
3.74 CUDBGWarpResources . . . 257
3.75 CUDBGWarpState . . . 258
3.76 CUDBGWarpState120 . . . 259
3.77 CUDBGWarpState127 . . . 259
3.78 CUDBGWarpState60 . . . 260
3.79 CuDim2 . . . 261
3.80 CuDim3 . . . 261

4 Data Fields **267**
4.1 A . . . 267
4.2 B . . . 268
4.3 C . . . 268
4.4 D . . . 270
4.5 E . . . 272
4.6 F . . . 273
4.7 G . . . 274
4.8 H . . . 277
4.9 I . . . 277
4.10 K . . . 278
4.11 L . . . 278
4.12 M . . . 279
4.13 N . . . 280
4.14 O . . . 280
4.15 P . . . 280
4.16 R . . . 281
4.17 S . . . 284
4.18 T . . . 285
4.19 U . . . 287
4.20 V . . . 287
4.21 W . . . 288
4.22 X . . . 289
4.23 Y . . . 289
4.24 Z . . . 289

5 Release Notes **291**
5.1 CUDA 13.4 Release . . . 291
5.1.1 Deprecated APIs . . . 291
5.1.2 Dropped APIs . . . 291
5.2 CUDA 13.3 Release . . . 291
5.2.1 New APIs . . . 291
5.3 CUDA 13.2 Release . . . 292
5.3.1 Major changes . . . 292
5.3.2 New APIs . . . 292
5.3.3 Deprecated APIs . . . 292
5.4 CUDA 13.1 Release . . . 292

5.4.1 Major changes . . . 292
5.4.2 New APIs . . . 293
5.4.3 Deprecated APIs . . . 293
5.4.4 Dropped APIs . . . 293
5.5 CUDA 13.0 Release . . . 293
5.5.1 Major changes . . . 293
5.5.2 New APIs . . . 294
5.5.3 Deprecated APIs . . . 294
5.6 CUDA 12.9 Release . . . 294
5.6.1 Major changes . . . 294
5.6.2 New APIs . . . 294
5.6.3 Updated APIs . . . 295
5.7 CUDA 12.8 Release . . . 295
5.7.1 Major changes . . . 295
5.7.2 New APIs . . . 295
5.8 CUDA 12.7 Release . . . 295
5.8.1 Major changes . . . 295
5.8.2 New APIs . . . 295
5.8.3 Updated APIs . . . 296
5.9 CUDA 12.6 Release . . . 296
5.9.1 Major changes . . . 296
5.9.2 New APIs . . . 296
5.10 CUDA 12.5 Release . . . 296
5.10.1 Major changes . . . 296
5.10.2 New APIs . . . 297
5.10.3 Deprecated APIs . . . 297
5.11 CUDA 12.4 Release . . . 297
5.11.1 Major changes . . . 297
5.11.2 New APIs . . . 297
5.11.3 Updated APIs . . . 298
5.12 CUDA 12.3 Release . . . 298
5.12.1 Major changes . . . 298
5.12.2 New APIs . . . 298
5.12.3 Updated APIs . . . 298
5.12.4 Deprecated APIs . . . 298
5.13 CUDA 12.2 Release . . . 299
5.13.1 Major changes . . . 299
5.13.2 New APIs . . . 299
5.14 CUDA 12.1 Release . . . 299
5.14.1 Major changes . . . 299
5.15 CUDA 12.0 Release . . . 299
5.15.1 Major changes . . . 299
5.15.2 New APIs . . . 300
5.15.3 Updated APIs . . . 300
5.16 CUDA 11.8 Release . . . 300
5.17 CUDA 7.0 Release . . . 301
5.18 CUDA 6.5 Release . . . 301

6 Notices **303**
6.1 Notice . . . 303
6.2 OpenCL . . . 304
6.3 Trademarks . . . 304

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.

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

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

CUDBG_ERROR_RECURSIVE_API_CALL when the call is made from within the notification callback set using CUDBGAPI_st::setNotifyNewEventCallback().

1.2. ELF and DWARF

CUDA applications are compiled in ELF binary format.

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

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

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

For memory reads, see:

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

For memory writes, see:

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

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

- `CUDBGAPI_st::readVirtualPC()`

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

```
<2><321>: Abbrev Number: 18 (DW_TAG_formal_parameter)
     DW_AT_decl_file   : 27
     DW_AT_decl_line   : 5
```

(continues on next page)

(continued from previous page)

```
     DW_AT_name        : res
     DW_AT_type        : <2c6>
     DW_AT_location    : 9 byte block: 3 18 0 0 0 0 0 0 0       (DW_OP_addr: 18)
     DW_AT_address_class: 7
```

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

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

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

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

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

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

- `CUDBGAPI_st::readPC()`

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).

1.5. Attaching and Detaching

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

1. Attach to the CPU process corresponding to the CUDA application. The CPU part of the application will be frozen at this point.
2. Check to see if the `CUDBG_DEBUGGER_INITIALIZED` variable is accessible from the memory space of the application. If not, it implies that the application has not loaded the CUDA driver, and the attaching to the application is complete.
3. If the `CUDBG_DEBUGGER_INITIALIZED` variable is accessible and non-zero, then the attach procedure is finished.
4. Check the existence and value of the global variable `CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD`:
    1. If it doesn't exist, then proceed with "Legacy debugger initialization" described below.
    2. On Linux, if it's less than zero, then the CUDA driver has not fully initialized yet. The attach process should be retried later, once this variable is greater than or equal to zero.
    3. If its value is greater than or equal to zero, then proceed with debugger initialization as described below.
5. Legacy debugger initialization: If the `CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD` variable is not present or is less than zero:

a. Make a dynamic (inferior) function call to the function `cudbgApiAttach()`, e.g. by using `ptrace(2)` on Linux. This causes a helper process to be forked off from the application, which assists in attaching to the CUDA process.

b. Ensure that the initialization of the CUDA debug API is complete, or wait till API initialization is successful (i.e. call the `initialize()` API method until it succeeds).

c. Make the `initializeAttachStub()` API call to initialize the helper process that was forked off from the application earlier.

1. If the `CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD` variable is greater than or equal to zero:

1. Insert a breakpoint in the `CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED` function.

2. Open the application's file descriptor whose number is given in `CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD` variable and write a single byte to it.

   On Linux, this can be achieved by using the `/proc` filesystem and the fd file `/proc/PID/FD`.

3. Resume all the threads of the attached program.
4. Wait until the `CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED` breakpoint is hit, remove the breakpoint and then stop/freeze all of the threads again.
5. Call the `initialize()` API method. It should now succeed.

1. Read the value of the `CUDBG_RESUME_FOR_ATTACH_DETACH` variable from the memory space of the application:
   - If the value is non-zero, resume the CUDA application so that more data can be collected about the application and sent to the debugger. When the application is resumed, the debug client can expect to receive various CUDA events from the CUDA application. Once all state has been collected, the debug client will receive the event `CUDBG_EVENT_ATTACH_COMPLETE`.
   - If the value is zero, there is no more attach data to collect. Set the `CUDBG_IPC_FLAG_NAME` variable to 1 in the application's process space, which enables further events from the CUDA application.
2. At this point, attaching to the CUDA application is complete and all GPUs belonging to the CUDA application will be suspended.

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

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

Chapter 2. APIs

2.1. Breakpoints

Macros

CUDBG_BREAKPOINT_HANDLE_ALL_USER_BREAKPOINTS
Special breakpoint handle - represents all non-special breakpoints that were inserted by the API user.

CUDBG_BREAKPOINT_HANDLE_BREAK_ON_LAUNCH
Break On Launch breakpoint handle - this breakpoint can be enabled/disabled with enableBreakpoint/disableBreakpoint.

CUDBG_BREAKPOINT_HANDLE_INTERNAL
Special breakpoint handle - internal breakpoint (e.g.

CUDBG_BREAKPOINT_HANDLE_INVALID
Invalid breakpoint handle.

CUDBG_BREAKPOINT_HANDLE_LEGACY
Special breakpoint handle - breakpoint inserted with setBreakpoint (handle-less).

CUDBG_BREAKPOINT_HANDLE_TRAP
Special breakpoint handle - hard-coded trap.

Enumerations

CUDBGAdjAddrAction
Describes which adjusted code address is to be returned.

Typedefs

CUDBGBreakpointHandle
GPU debugger breakpoint handle.

Variables

CUDBGResult(* *CUDBGAPI_st::disableBreakpoint*
Disable a breakpoint specified by its handle.

CUDBGResult(* *CUDBGAPI_st::enableBreakpoint*
Enable a breakpoint specified by its handle.

CUDBGResult(* *CUDBGAPI_st::getAdjustedCodeAddress*
Get the adjusted code address for a given code address for a given device.

CUDBGResult(* *CUDBGAPI_st::getWarpHitBreakpoint*
Get the handle of the breakpoint that the given warp hit.

CUDBGResult(* *CUDBGAPI_st::insertBreakpoint*
Set a breakpoint at the given instruction address for the given device.

CUDBGResult(* *CUDBGAPI_st::isBreakpointEnabled*
Check if a breakpoint specified by its handle is enabled.

CUDBGResult(* *CUDBGAPI_st::removeBreakpoint*
Remove a breakpoint specified by its handle.

CUDBGResult(* *CUDBGAPI_st::setBreakpoint*
Set a breakpoint at the given instruction address for the given device.

CUDBGResult(* *CUDBGAPI_st::setBreakpoint31*
Set a breakpoint at the given instruction address.

CUDBGResult(* *CUDBGAPI_st::unsetBreakpoint*
Unset a breakpoint at the given instruction address for the given device.

CUDBGResult(* *CUDBGAPI_st::unsetBreakpoint31*
Unset a breakpoint at the given instruction address.

2.1.1. Macros

`CUDBG_BREAKPOINT_HANDLE_ALL_USER_BREAKPOINTS` (0xFFFFFFFFFFFFFFFBULL)

Special breakpoint handle - represents all non-special breakpoints that were inserted by the API user.

This handle can be used to enable, disable, and remove all added breakpoints in a single call.

It can only be used as a handle in *CUDBGAPI_st::removeBreakpoint*, *CUDBGAPI_st::enableBreakpoint*, *CUDBGAPI_st::disableBreakpoint* functions.

If this handle is used and there are no breakpoints added by the user, the aforementioned API functions do not fail, instead the operation acts on no breakpoints, so calling e.g. *CUDBGAPI_st::removeBreakpoint* with this handle is fine if there are no breakpoints added by the user. They may still fail for other reasons.

Note: The set of breakpoints represented by this handle does not include the CUDBG_BREAKPOINT_HANDLE_BREAK_ON_LAUNCH breakpoint.

CUDBG_BREAKPOINT_HANDLE_BREAK_ON_LAUNCH (0xFFFFFFFFFFFFFFFCULL)

Break On Launch breakpoint handle - this breakpoint can be enabled/disabled with enableBreakpoint/disableBreakpoint.

It is hit once every time a kernel is launched. It can be hit by one or more warps, but only once. Only supports CUDA Kernel launches (both host-side and device-side launches).

CUDBG_BREAKPOINT_HANDLE_INTERNAL (0xFFFFFFFFFFFFFFFDULL)

Special breakpoint handle - internal breakpoint (e.g.

used internally for stepping)

CUDBG_BREAKPOINT_HANDLE_INVALID (0ULL)

Invalid breakpoint handle.

CUDBG_BREAKPOINT_HANDLE_LEGACY (0xFFFFFFFFFFFFFFFEULL)

Special breakpoint handle - breakpoint inserted with setBreakpoint (handle-less).

CUDBG_BREAKPOINT_HANDLE_TRAP (0xFFFFFFFFFFFFFFFFULL)

Special breakpoint handle - hard-coded trap.

2.1.2. Enumerations

enum **CUDBGAdjAddrAction**

Describes which adjusted code address is to be returned.

Values:

enumerator **CUDBG_ADJ_PREVIOUS_ADDRESS** = 0x000

Get the adjusted previous code address.

enumerator **CUDBG_ADJ_CURRENT_ADDRESS** = 0x001

Get the adjusted next code address.

enumerator **CUDBG_ADJ_NEXT_ADDRESS** = 0x002

Get the adjusted current code address.

2.1.3. Typedefs

typedef uint64_t **CUDBGBreakpointHandle**

GPU debugger breakpoint handle.

Valid breakpoint handles will grow from 1. Handle values of removed breakpoints can be reused in some cases. It is not guaranteed that new (non-reused) breakpoint handles will always be increasing by 1, there can be gaps. A valid handle always corresponds to a single inserted breakpoint,

and each breakpoint can only have one handle. Handles with the highest bit set are reserved for special purposes and can correspond to multiple breakpoints.

2.2. DWARF Utilities

Enumerations

CUDBGElfImageType
CUDA ELF image type.

CUDBGRegClass
Physical register types.

Variables

CUDBGResult(* *CUDBGAPI_st::disassemble*
Disassemble instruction at instruction address.

CUDBGResult(* *CUDBGAPI_st::getElfImageByHandle*
Get the relocated or non-relocated ELF image for the given handle on the given device.

CUDBGResult(* *CUDBGAPI_st::getHostAddrFromDeviceAddr*
Given a device virtual address, return a corresponding system memory virtual address.

CUDBGResult(* *CUDBGAPI_st::getPhysicalRegister30*
Get the physical register number(s) assigned to a virtual register name at a given PC, if it's live at that PC.

CUDBGResult(* *CUDBGAPI_st::getPhysicalRegister40*
Get the physical register number(s) assigned to a virtual register name at a given PC, if it's live at that PC.

CUDBGResult(* *CUDBGAPI_st::isDeviceCodeAddress*
Determine whether a virtual address resides within device code.

CUDBGResult(* *CUDBGAPI_st::isDeviceCodeAddress55*
Determine whether a virtual address resides within device code.

CUDBGResult(* *CUDBGAPI_st::lookupDeviceCodeSymbol*
Determines whether a symbol represents a function in device code and returns its virtual address.

2.2.1. Enumerations

enum **CUDBGElfImageType**
CUDA ELF image type.

Values:

enumerator **CUDBG_ELF_IMAGE_TYPE_NONRELOCATED** = 0
Non-relocated ELF image.

enumerator **CUDBG_ELF_IMAGE_TYPE_RELOCATED** = 1

Relocated ELF image.

enum **CUDBGRegClass**

Physical register types.

Values:

enumerator **REG_CLASS_INVALID** = 0x000

The physical register is invalid.

enumerator **REG_CLASS_REG_CC** = 0x001

The physical register is a condition code register.

Unused.

enumerator **REG_CLASS_REG_PRED** = 0x002

The physical register is a predicate register.

Unused.

enumerator **REG_CLASS_REG_ADDR** = 0x003

The physical register is an address register.

Unused.

enumerator **REG_CLASS_REG_HALF** = 0x004

The physical register is a 16-bit register.

Unused.

enumerator **REG_CLASS_REG_FULL** = 0x005

The physical register is a 32-bit register.

enumerator **REG_CLASS_MEM_LOCAL** = 0x006

The content of the physical register has been spilled to memory.

enumerator **REG_CLASS_LMEM_REG_OFFSET** = 0x007

The content of the physical register has been spilled to the local stack (ABI only).

enumerator **REG_CLASS_UREG_PRED** = 0x009

The physical register is a uniform predicate register.

enumerator **REG_CLASS_UREG_HALF** = 0x00a

The physical register is a 16-bit uniform register.

enumerator **REG_CLASS_UREG_FULL** = 0x00b

The physical register is a 32-bit uniform register.

enumerator **REG_CLASS_TEMP_REG_SPILL** = 0x00c

The physical register is a temp register spill.

Temp register spill 0 is RPC.LO and 1 is RPC.HI.

2.3. Device Execution Control

Enumerations

CUDBGSingleStepFlags
Single step flags.

CUDBGSingleStepType
Single step operation type (API method that was used to start the single step operation)

Variables

CUDBGResult(* *CUDBGAPI_st::executeInternalCommand*
Execute an internal command (not available in public driver builds)

CUDBGResult(* *CUDBGAPI_st::resumeAllDevices*
Resume all running CUDA devices.

CUDBGResult(* *CUDBGAPI_st::resumeDevice*
Resume a suspended CUDA device.

CUDBGResult(* *CUDBGAPI_st::resumeWarpsUntilPC*
Insert a temporary breakpoint at the specified virtual PC and resume all warps in the specified bitmask on a given SM.

CUDBGResult(* *CUDBGAPI_st::resumeWarpsUntilPC60*
Insert a temporary breakpoint at the specified virtual PC and resume all warps in the specified bitmask on a given SM.

CUDBGResult(* *CUDBGAPI_st::setKernelLaunchNotificationMode*
Set the launch notification policy.

CUDBGResult(* *CUDBGAPI_st::singleStepWarp*
Single step an individual warp nsteps times on a suspended CUDA device.

CUDBGResult(* *CUDBGAPI_st::singleStepWarp40*
Single step an individual warp on a suspended CUDA device.

CUDBGResult(* *CUDBGAPI_st::singleStepWarp41*
Single step an individual warp on a suspended CUDA device.

CUDBGResult(* *CUDBGAPI_st::singleStepWarp65*
Single step an individual warp nsteps times on a suspended CUDA device.

CUDBGResult(* *CUDBGAPI_st::suspendAllDevices*
Suspend all running CUDA devices.

CUDBGResult(* *CUDBGAPI_st::suspendDevice*
Suspends a running CUDA device.

2.3.1. Enumerations

enum **CUDBGSingleStepFlags**

Single step flags.

Values:

enumerator **CUDBG_SINGLE_STEP_FLAGS_NONE** = 0

Default behavior.

enumerator **CUDBG_SINGLE_STEP_FLAGS_NO_STEP_OVER_WARP_BARRIERS** = (1U << 0)

Disable optimized warp barrier stepping.

Do not step over warp-wide barriers using a breakpoint and resume, instead perform a single step and return. Passing this flag in means that the API client plans to repeat the singleStepWarp() call until the warp barrier is stepped over. This gives a more precise exception information if an exception is encountered by the diverged threads while stepping. This flag is only valid for the singleStepWarp() API method, resumeWarpsUntilPC() always steps over warp-wide barriers.

enumerator **CUDBG_SINGLE_STEP_FLAGS_NON_BLOCKING** = (1U << 1)

Don't block on the stepping operations, instead return early and send an event once stepping is done.

When the step is completed, CUDBG_EVENT_SINGLE_STEP_COMPLETE is sent. If the step is interrupted by a breakpoint or an exception, CUDBG_EVENT_ALL_DEVICES_SUSPENDED is sent instead.

enum **CUDBGSingleStepType**

Single step operation type (API method that was used to start the single step operation)

Values:

enumerator **CUDBG_SINGLE_STEP_TYPE_INVALID** = 0

Invalid single step operation type.

enumerator **CUDBG_SINGLE_STEP_TYPE_SINGLE_STEP_WARP** = 1

The singleStepWarp() API method was used to start the single step operation.

enumerator **CUDBG_SINGLE_STEP_TYPE_RESUME_WARPS_UNTIL_PC** = 2

The resumeWarpsUntilPC() API method was used to start the single step operation.

2.4. Device Properties

Variables

CUDBGResult(* *CUDBGAPI_st::getDeviceName*
Get the device name string.

CUDBGResult(* *CUDBGAPI_st::getDeviceType*
Get the string description of the device.

CUDBGResult(* *CUDBGAPI_st::getNumDevices*
Get the number of installed CUDA devices.

CUDBGResult(* *CUDBGAPI_st::getNumLanes*
Get the number of lanes per warp on the device.

CUDBGResult(* *CUDBGAPI_st::getNumPredicates*
Get the number of predicate registers per lane on the device.

CUDBGResult(* *CUDBGAPI_st::getNumRegisters*
Get the maximum number of registers per lane on the device.

CUDBGResult(* *CUDBGAPI_st::getNumSMs*
Get the total number of SMs on the device.

CUDBGResult(* *CUDBGAPI_st::getNumUniformPredicates*
Get the number of uniform predicate registers per warp on the device.

CUDBGResult(* *CUDBGAPI_st::getNumUniformRegisters*
Get the number of uniform registers per warp on the device.

CUDBGResult(* *CUDBGAPI_st::getNumWarps*
Get the number of warps per SM on the device.

CUDBGResult(* *CUDBGAPI_st::getSmType*
Get the SM type of the device.

2.5. Device State Alteration

Variables

CUDBGResult(* *CUDBGAPI_st::writeCCRegister*
Write to the hardware CC register.

CUDBGResult(* *CUDBGAPI_st::writeGenericMemory*
Write to an address in any memory segment.

CUDBGResult(* *CUDBGAPI_st::writeGlobalMemory*
Write to an address in global memory.

CUDBGResult(* *CUDBGAPI_st::writeGlobalMemory31*
Write to an address in global memory.

CUDBGResult(* *CUDBGAPI_st::writeGlobalMemory55*
Write to an address in global memory.

CUDBGResult(* *CUDBGAPI_st::writeLocalMemory*
Write to an address in local memory.

CUDBGResult(* *CUDBGAPI_st::writeParamMemory*
Write to an address in param memory.

CUDBGResult(* *CUDBGAPI_st::writePinnedMemory*
Write to a pinned memory address.

CUDBGResult(* *CUDBGAPI_st::writePredicates*
Write to hardware predicates.

CUDBGResult(* *CUDBGAPI_st::writeRegister*
Write to a hardware register.

CUDBGResult(* *CUDBGAPI_st::writeRpcRegisters*
Write the RPC.LO and/or RPC.HI hardware registers for a specific active thread.

CUDBGResult(* *CUDBGAPI_st::writeSharedMemory*
Write to an address in shared memory.

CUDBGResult(* *CUDBGAPI_st::writeTcgen05TmemMemory*
Write tcgen05 Tensor Memory data for the CTA containing a warp.

CUDBGResult(* *CUDBGAPI_st::writeUniformPredicates*
Write to hardware uniform predicates.

CUDBGResult(* *CUDBGAPI_st::writeUniformRegister*
Write to a hardware uniform register.

2.6. Device State Inspection

Enumerations

CUDBGBarrierScope
CUDA barrier scope.

CUDBGCbuThreadState
Thread state in the CBU (Convergence Barrier Unit).

CUDBGCoredumpGenerationFlags
Coredump generation flags.

CUDBGCudaLogLevel
CUDA Log severity level.

CUDBGCudaLogLevelFilter
CUDA Log level filter.

CUDBGCudaLogRuleAction
CUDA Log filtering rule action.

CUDBGDeviceInfoAttribute_t
Device-level attributes.

CUDBGDeviceInfoQueryType_t
Device info query type.

CUDBGLaneInfoAttribute_t
Lane-level (thread-level) attributes.

CUDBGSMInfoAttribute_t
SM-level attributes.

CUDBGTcgen05TmemAllocationPhase
Tensor Memory allocation phase.

CUDBGWarpInfoAttribute_t
Warp-level attributes.

ptxStorageKind
Memory segments for DWARF.

Structs

CUDBGCbuWarpState
Warp state in the CBU (Convergence Barrier Unit).

CUDBGCudaLogMessage
CUDA Log Message.

CUDBGCudaLogMessage129
CUDA Log Message for API version 12.9.

CUDBGCudaLogRule
CUDA Log filtering rule.

CUDBGDeviceInfo
Device-level information.

CUDBGDeviceInfoSizes
Sizes of the various structs returned by the batched device update APIs.

CUDBGLaneState
Lane state (state of a single thread).

CUDBGLoadedFunctionInfo
Information about a lazily loaded function.

CUDBGMemoryInfo
Memory information.

CUDBGSMInfo
SM-level information.

CUDBGTcgen05TmemInfo
tcgen05 Tensor Memory information for a warp's CTA.

CUDBGWarpInfo
Warp-level information.

CUDBGWarpResources
Warp resources.

CUDBGWarpState
Warp state information.

CUDBGWarpState120
Warp state for API version 12.0.

CUDBGWarpState127
Warp state for API version 12.7.

CUDBGWarpState60
Warp state for API version 6.0.

Variables

CUDBGResult(* *CUDBGAPI_st::consumeCudaLogs*
Get CUDA error log entries.

CUDBGResult(* *CUDBGAPI_st::consumeCudaLogs129*
Get CUDA error log entries.

CUDBGResult(* *CUDBGAPI_st::generateCoredump*
Generate a coredump for the current GPU state.

CUDBGResult(* *CUDBGAPI_st::getBindlessConstAddress*
Get bindless constant GPU VA and size from its header.

CUDBGResult(* *CUDBGAPI_st::getCbuWarpState*
Get the CBU state of a given warp.

CUDBGResult(* *CUDBGAPI_st::getClusterExceptionTargetBlock*
Get the target block index and validity status for cluster exceptions.

CUDBGResult(* *CUDBGAPI_st::getConstBankAddress*
Get constant bank GPU VA and size.

CUDBGResult(* *CUDBGAPI_st::getConstBankAddress123*
Convert constant bank number and offset into GPU VA.

CUDBGResult(* *CUDBGAPI_st::getCudaExceptionString*
Get the error string for CUDA Exceptions.

CUDBGResult(* *CUDBGAPI_st::getDeviceInfo*
Read full device info for the device.

CUDBGResult(* *CUDBGAPI_st::getDeviceInfoSizes*
Return sizes for device info structs and defined attributes.

CUDBGResult(* *CUDBGAPI_st::getDevicePCIBusInfo*
Get PCI bus and device ids associated with device index.

CUDBGResult(* *CUDBGAPI_st::getHardwareBarrierInfo*
Get hardware barrier information.

CUDBGResult(* *CUDBGAPI_st::getLoadedFunctionInfo*
Get the section number and address of loaded functions for a given module.

CUDBGResult(* *CUDBGAPI_st::getLoadedFunctionInfo118*
Get the section number and address of loaded functions for a given module.

CUDBGResult(* *CUDBGAPI_st::getManagedMemoryRegionInfo*
Get a sorted list of managed memory regions.

CUDBGResult(* *CUDBGAPI_st::memcheckReadErrorAddress*
Get the address that memcheck detected an error on.

CUDBGResult(* *CUDBGAPI_st::readActiveLanes*
Read the lane bitmask of active threads on a valid warp.

CUDBGResult(* *CUDBGAPI_st::readAllVirtualReturnAddresses*
Read all the virtual return addresses for a thread (the full backtrace).

CUDBGResult(* *CUDBGAPI_st::readBlockIdx*
Read the CUDA block index running on a valid warp.

CUDBGResult(* *CUDBGAPI_st::readBlockIdx32*
Read the two-dimensional CUDA block index running on a valid warp.

CUDBGResult(* *CUDBGAPI_st::readBrokenWarps*
Read the bitmask of warps that are at a breakpoint on a given SM.

CUDBGResult(* *CUDBGAPI_st::readCCRegister*
Read the hardware CC register.

CUDBGResult(* *CUDBGAPI_st::readCPUCallStack*
Read the CPU call stack captured at the time of kernel launch.

CUDBGResult(* *CUDBGAPI_st::readCallDepth*
Read the call depth (number of calls) for a given thread.

CUDBGResult(* *CUDBGAPI_st::readCallDepth32*
Read the call depth (number of calls) for a given warp.

CUDBGResult(* *CUDBGAPI_st::readClusterIdx*
Read the CUDA cluster index running on a valid warp.

CUDBGResult(* *CUDBGAPI_st::readCodeMemory*
Read content at address in the code memory segment.

CUDBGResult(* *CUDBGAPI_st::readConstMemory129*
Read content at address in the constant memory segment.

CUDBGResult(* *CUDBGAPI_st::readDeviceExceptionState*
Get the exception state of the SMs on the device.

CUDBGResult(* *CUDBGAPI_st::readDeviceExceptionState80*
Get the exception state of the SMs on the device.

CUDBGResult(* *CUDBGAPI_st::readErrorPC*
Get the hardware reported error PC if it exists.

CUDBGResult(* *CUDBGAPI_st::readGenericMemory*
Read content at an address in any memory segment.

CUDBGResult(* *CUDBGAPI_st::readGlobalMemory*
Read content at an address in the global address space.

CUDBGResult(* *CUDBGAPI_st::readGlobalMemory31*
Read content at address in the global memory segment.

CUDBGResult(* *CUDBGAPI_st::readGlobalMemory55*
Read content at address in the global memory segment.

CUDBGResult(* *CUDBGAPI_st::readGridId*
Read the 64-bit CUDA grid index running on a valid warp.

CUDBGResult(* *CUDBGAPI_st::readGridId50*
Read the CUDA grid index running on a valid warp.

CUDBGResult(* *CUDBGAPI_st::readLaneException*
Read the exception type for a given thread.

CUDBGResult(* *CUDBGAPI_st::readLaneStatus*
Read the status of the given thread.

CUDBGResult(* *CUDBGAPI_st::readLocalMemory*
Read content at address in the local memory segment.

CUDBGResult(* *CUDBGAPI_st::readPC*
Read the PC offset on the given active thread.

CUDBGResult(* *CUDBGAPI_st::readParamMemory*
Read content at address in the param memory segment.

CUDBGResult(* *CUDBGAPI_st::readPinnedMemory*
Read content at pinned address in system memory.

CUDBGResult(* *CUDBGAPI_st::readPredicates*
Read content of hardware predicate registers.

CUDBGResult(* *CUDBGAPI_st::readRegister*
Read content of a hardware register.

CUDBGResult(* *CUDBGAPI_st::readRegisterRange*
Read content of a hardware range of hardware registers.

CUDBGResult(* *CUDBGAPI_st::readRegisterRange60*
Read content of a range of hardware registers.

CUDBGResult(* *CUDBGAPI_st::readReturnAddress*
Read the return address (offset) for a call level.

CUDBGResult(* *CUDBGAPI_st::readReturnAddress32*
Read the return address (offset) for a call level.

CUDBGResult(* *CUDBGAPI_st::readRpcRegisters*
Read the RPC.LO and/or RPC.HI hardware registers for a specific thread.

CUDBGResult(* *CUDBGAPI_st::readSharedMemory*
Read content at address in the shared memory segment.

CUDBGResult(* *CUDBGAPI_st::readSmException*
Get the SM exception status if it exists.

CUDBGResult(* *CUDBGAPI_st::readSyscallCallDepth*
Read the call depth of syscalls for a given thread.

CUDBGResult(* *CUDBGAPI_st::readTcgen05TmemInfo*
Read tcgen05 Tensor Memory metadata for the CTA containing a warp.

CUDBGResult(* *CUDBGAPI_st::readTcgen05TmemMemory*
Read tcgen05 Tensor Memory data for the CTA containing a warp.

CUDBGResult(* *CUDBGAPI_st::readTextureMemory*
This method is no longer supported since CUDA 12.0.

CUDBGResult(* *CUDBGAPI_st::readTextureMemoryBindless*
This method is no longer supported since CUDA 12.0.

CUDBGResult(* *CUDBGAPI_st::readThreadIdx*
Read the CUDA thread index running on valid thread.

CUDBGResult(* *CUDBGAPI_st::readUniformPredicates*
Read contents of uniform predicate registers.

CUDBGResult(* *CUDBGAPI_st::readUniformRegisterRange*
Read a range of uniform registers.

CUDBGResult(* *CUDBGAPI_st::readValidLanes*
Read the lane bitmask of valid threads on a given warp.

CUDBGResult(* *CUDBGAPI_st::readValidWarps*
Read the bitmask of valid warps on a given SM.

CUDBGResult(* *CUDBGAPI_st::readVirtualPC*
Read the PC (virtual address) on the given active thread.

CUDBGResult(* *CUDBGAPI_st::readVirtualReturnAddress*
Read the virtual return address for a call level.

CUDBGResult(* *CUDBGAPI_st::readVirtualReturnAddress32*
Read the virtual return address for a call level.

CUDBGResult(* *CUDBGAPI_st::readWarpResources*
Get the resources assigned to a given warp.

CUDBGResult(* *CUDBGAPI_st::readWarpState*
Read the state of a given warp.

CUDBGResult(* *CUDBGAPI_st::readWarpState120*
Read the state of a given warp.

CUDBGResult(* *CUDBGAPI_st::readWarpState127*
Read the state of a given warp.

CUDBGResult(* *CUDBGAPI_st::readWarpState60*
Read the state of a given warp.

CUDBGResult(* *CUDBGAPI_st::setCudaLogRules*
Set CUDA log filtering rules.

2.6.1. Enumerations

enum **CUDBGBarrierScope**

CUDA barrier scope.

Values:

enumerator **CUDBG_BARRIER_SCOPE_INVALID** = 0xFFFFFFFFU

Invalid barrier scope.

enumerator **CUDBG_BARRIER_SCOPE_NONE** = 0

No barrier.

enumerator **CUDBG_BARRIER_SCOPE_WARP** = 1

Warp-wide barrier.

enumerator **CUDBG_BARRIER_SCOPE_WARP_GROUP** = 2

Warp group-wide barrier.

enumerator **CUDBG_BARRIER_SCOPE_BLOCK** = 3
Block-wide barrier.

enumerator **CUDBG_BARRIER_SCOPE_CLUSTER** = 4
Cluster-wide barrier.

enumerator **CUDBG_BARRIER_SCOPE_KERNEL** = 5
Kernel-wide barrier.

enum **CUDBGCbuThreadState**
Thread state in the CBU (Convergence Barrier Unit).
Values:

enumerator **CUDBG_CBU_THREAD_STATE_INVALID** = 0xFFFFFFFFU

enumerator **CUDBG_CBU_THREAD_STATE_EXITED** = 0

enumerator **CUDBG_CBU_THREAD_STATE_READY**

enumerator **CUDBG_CBU_THREAD_STATE_YIELDED**

enumerator **CUDBG_CBU_THREAD_STATE_SLEEP**

enumerator **CUDBG_CBU_THREAD_STATE_SLEEPYIELD**

enumerator **CUDBG_CBU_THREAD_STATE_READYATNEXT**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDPLUS**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDALL**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDCOLLECTIVE**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB0**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB1**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB2**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB3**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB4**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB5**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB6**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB7**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB8**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB9**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB10**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB11**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB12**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB13**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB14**

enumerator **CUDBG_CBU_THREAD_STATE_BLOCKEDB15**

enum **CUDBGCoredumpGenerationFlags**

Coredump generation flags.

Values:

enumerator **CUDBG_COREDUMP_DEFAULT_FLAGS** = 0

Default flags.

enumerator **CUDBG_COREDUMP_SKIP_NONRELOCATED_ELF_IMAGES** = (1 << 0)

Skip dumping non-relocated ELF images.

enumerator **CUDBG_COREDUMP_SKIP_GLOBAL_MEMORY** = (1 << 1)

Skip dumping global memory.

enumerator **CUDBG_COREDUMP_SKIP_SHARED_MEMORY** = (1 << 2)

Skip dumping shared memory.

enumerator **CUDBG_COREDUMP_SKIP_LOCAL_MEMORY** = (1 << 3)

Skip dumping local memory.

enumerator **CUDBG_COREDUMP_SKIP_CONSTBANK_MEMORY** = (1 << 5)

Skip dumping constant bank memory.

enumerator **CUDBG_COREDUMP_GZIP_COMPRESS** = (1 << 6)

Compress the coredump with gzip.

enumerator **CUDBG_COREDUMP_FAULTED_CONTEXTS_ONLY** = (1 << 7)

Only include data for contexts which have encountered an exception.

If this flag is used and there are no faulted contexts on any device, then generateCoredump() method will return CUDBG_ERROR_INVALID_CONTEXT error code. Contexts that have warps at breakpoints count as faulted.

enumerator **CUDBG_COREDUMP_SKIP_TENSOR_MEMORY** = (1 << 8)

Skip dumping tensor memory.

enumerator **CUDBG_COREDUMP_LIGHTWEIGHT_FLAGS** = *CUDBG_COREDUMP_SKIP_NONRELOCATED_ELF_IMAGES* | *CUDBG_COREDUMP_SKIP_GLOBAL_MEMORY* | *CUDBG_COREDUMP_SKIP_SHARED_MEMORY* | *CUDBG_COREDUMP_SKIP_LOCAL_MEMORY* | *CUDBG_COREDUMP_SKIP_CONSTBANK_MEMORY* | *CUDBG_COREDUMP_SKIP_TENSOR_MEMORY*

Lightweight flags.

enum **CUDBGCudaLogLevel**

CUDA Log severity level.

Values:

enumerator **CUDBG_CUDA_LOG_LEVEL_INVALID** = 0xFFFFFFFFU

Invalid log level.

enumerator **CUDBG_CUDA_LOG_LEVEL_ERROR** = 0

Error log level, matches CU_LOG_LEVEL_ERROR.

enumerator **CUDBG_CUDA_LOG_LEVEL_WARNING** = 1

Warning log level, matches CU_LOG_LEVEL_WARNING.

enum **CUDBGCudaLogLevelFilter**

CUDA Log level filter.

Values:

enumerator **CUDBG_CUDA_LOG_LEVEL_FILTER_ANY** = 0xFFFFFFFFU

Any log level.

enumerator **CUDBG_CUDA_LOG_LEVEL_FILTER_ERROR** = 0

Error log level.

enumerator **CUDBG_CUDA_LOG_LEVEL_FILTER_WARNING** = 1

Warning log level.

enum **CUDBGCudaLogRuleAction**

CUDA Log filtering rule action.

Values:

enumerator **CUDBG_CUDA_LOG_RULE_ACTION_EXCLUDE** = 0xFFFFFFFFU

Exclude the log message.

enumerator **CUDBG_CUDA_LOG_RULE_ACTION_SEND_ASYNC** = 0

Send the log message asynchronously.

enumerator **CUDBG_CUDA_LOG_RULE_ACTION_SEND_SYNC** = 1

Send the log message synchronously.

enum **CUDBGDeviceInfoAttribute_t**

Device-level attributes.

Values:

enumerator **CUDBG_DEVICE_ATTRIBUTE_SM_UPDATE_MASK** = 0

Mask of updated SMs reported by this response.

Optional: Yes, assume all 1's if absent. Size: Number of SMs-sized bitmask, rounded up to be divisible by 8.

enumerator **CUDBG_DEVICE_ATTRIBUTE_SM_ACTIVE_MASK** = 1

Mask of SMs with any valid warp.

Optional: No, always returned by the API. Size: Number of SMs-sized bitmask, rounded up to be divisible by 8.

enumerator **CUDBG_DEVICE_ATTRIBUTE_SM_EXCEPTION_MASK** = 2

Mask of SMs with any warps with exceptions.

Optional: Yes, assume all 0's if absent. Size: Number of SMs-sized bitmask, rounded up to be divisible by 8.

enumerator **CUDBG_DEVICE_ATTRIBUTE_COUNT**

Device attributes count.

enum **CUDBGDeviceInfoQueryType_t**

Device info query type.

Values:

enumerator **CUDBG_RESPONSE_TYPE_FULL**

Request state information for all valid SMs/Warps/Lanes.

enumerator **CUDBG_RESPONSE_TYPE_UPDATE**

Request state information for all changed SMs/Warps/Lanes since the last call.

It's safe to always use this type, the API will respond with the full information when necessary.

enumerator **CUDBG_RESPONSE_TYPE_UNKNOWN** = 0xFFFFFFFFU

Unknown device info query type.

enum **CUDBGLaneInfoAttribute_t**

Lane-level (thread-level) attributes.

Values:

enumerator **CUDBG_LANE_ATTRIBUTE_COUNT**

Lane (thread) attributes count.

enum **CUDBGSMInfoAttribute_t**

SM-level attributes.

Values:

enumerator **CUDBG_SM_ATTRIBUTE_WARP_UPDATE_MASK** = 0

Mask of updated warps reported by this response.

Optional: Yes, assume all 1's if absent. Size: uint64_t.

enumerator **CUDBG_SM_ATTRIBUTE_COUNT**

SM attributes count.

enum **CUDBGTcgen05TmemAllocationPhase**

Tensor Memory allocation phase.

Values:

enumerator **CUDBG_TCGEN05_TMEM_ALLOCATION_PHASE_UNKNOWN** = 0

Tensor Memory allocation phase is unknown to this API version.

enumerator **CUDBG_TCGEN05_TMEM_ALLOCATION_PHASE_ALLOC_PERMIT_AVAILABLE** = 1

Tensor Memory allocation permit is available.

enumerator **CUDBG_TCGEN05_TMEM_ALLOCATION_PHASE_ALLOC_PERMIT_RELINQUISHED** = 2

Tensor Memory allocation permit has been relinquished.

enumerator **CUDBG_TCGEN05_TMEM_ALLOCATION_PHASE_EXCLUSIVE_ALLOC_ACTIVE** = 3

Tensor Memory allocation permit has been claimed by an exclusive allocation.

enum **CUDBGWarpInfoAttribute_t**

Warp-level attributes.

Values:

enumerator **CUDBG_WARP_ATTRIBUTE_LANE_UPDATE_MASK** = 0

Mask of updated lanes reported by this response.

Optional: Yes, assume all 1's if absent. Size: uint32_t.

enumerator **CUDBG_WARP_ATTRIBUTE_LANE_ATTRIBUTES** = 1

Signals whether the attribute flags field is present on the lane level for this warp.

Optional: Yes, assume no lane attributes for this warp if absent. Size: 0 (doesn't have an associated warp-level field).

enumerator **CUDBG_WARP_ATTRIBUTE_EXCEPTION** = 2

CUDBGException_t for this warp.

Optional: Yes, assume CUDBG_EXCEPTION_NONE if absent. Size: uint32_t.

enumerator **CUDBG_WARP_ATTRIBUTE_ERRORPC** = 3

Error PC for this warp.

Optional: Yes, assume no error PC is available if absent. Size: uint64_t.

enumerator **CUDBG_WARP_ATTRIBUTE_CLUSTERIDX** = 4

Cluster index for this warp.

Optional: Yes if warp is not in a cluster. Size: *CuDim3*.

enumerator **CUDBG_WARP_ATTRIBUTE_CLUSTERDIM** = 5

Cluster dimensions for this warp.

Optional: Yes if warp is not in a cluster. Size: *CuDim3*.

enumerator **CUDBG_WARP_ATTRIBUTE_CLUSTER_EXCEPTION_TARGET_BLOCK_IDX** = 6

For cluster exceptions, this represents the target block index handling cluster requests.

Optional: Yes, assume no block index is available if absent. Size: *CuDim3*.

enumerator **CUDBG_WARP_ATTRIBUTE_IN_SYSCALL_LANES** = 7

Lane mask showing threads that are in a syscall.

Optional: Yes, use readSyscallCallDepth() if this attribute is not present. Size: uint32_t.

enumerator **CUDBG_WARP_ATTRIBUTE_HIT_BREAKPOINT_HANDLE** = 8

Breakpoint handle of a broken warp (of type CUDBGBreakpointHandle) Optional: Yes, only present for broken warps Size: CUDBGBreakpointHandle.

enumerator **CUDBG_WARP_ATTRIBUTE_COUNT**

Warp attributes count.

enum **ptxStorageKind**

Memory segments for DWARF.

Values:

enumerator **ptxUNSPECIFIEDStorage** = 0x0

enumerator **ptxCodeStorage** = 0x1

enumerator **ptxRegStorage** = 0x2

enumerator **ptxSregStorage** = 0x3

enumerator **ptxConstStorage** = 0x4

enumerator **ptxGlobalStorage** = 0x5

enumerator **ptxLocalStorage** = 0x6

enumerator **ptxParamStorage** = 0x7

enumerator **ptxSharedStorage** = 0x8

enumerator **ptxSurfStorage** = 0x9

enumerator **ptxTexStorage** = 0xA

enumerator **ptxTexSamplerStorage** = 0xB

enumerator **ptxGenericStorage** = 0xC

enumerator **ptxIParamStorage** = 0xD

enumerator **ptxOParamStorage** = 0xE

enumerator **ptxFrameStorage** = 0xF

enumerator **ptxURegStorage** = 0x10

enumerator **ptxBoundConstStorage0** = 0x20

enumerator **ptxBoundConstStorage1** = 0x21

enumerator **ptxBoundConstStorage2** = 0x22

enumerator **ptxBoundConstStorage3** = 0x23

enumerator **ptxBoundConstStorage4** = 0x24

enumerator **ptxBoundConstStorage5** = 0x25

enumerator **ptxBoundConstStorage6** = 0x26

enumerator **ptxBoundConstStorage7** = 0x27

enumerator **ptxBoundConstStorage8** = 0x28

enumerator **ptxBoundConstStorage9** = 0x29

enumerator **ptxBoundConstStorage10** = 0x2A

enumerator **ptxBoundConstStorage11** = 0x2B

enumerator **ptxBoundConstStorage12** = 0x2C

enumerator **ptxBoundConstStorage13** = 0x2D

enumerator **ptxBoundConstStorage14** = 0x2E

enumerator **ptxBoundConstStorage15** = 0x2F

enumerator **ptxBoundConstStorage16** = 0x30

enumerator **ptxBoundConstStorage17** = 0x31

enumerator **ptxBoundConstStorage18** = 0x32

enumerator **ptxBoundConstStorage19** = 0x33

enumerator **ptxBoundConstStorage20** = 0x34

enumerator **ptxBoundConstStorage21** = 0x35

enumerator **ptxBoundConstStorage22** = 0x36

enumerator **ptxBoundConstStorage23** = 0x37

enumerator **ptxBoundConstStorage24** = 0x38

enumerator **ptxBoundConstStorage25** = 0x39

enumerator **ptxBoundConstStorage26** = 0x3A

enumerator **ptxBoundConstStorage27** = 0x3B

enumerator **ptxBoundConstStorage28** = 0x3C

enumerator **ptxBoundConstStorage29** = 0x3D

enumerator **ptxBoundConstStorage30** = 0x3E

enumerator **ptxBoundConstStorage31** = 0x3F

enumerator **ptxMAXStorage**

2.7. Events

One of those events will create a *CUDBGEvent*:

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

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;
```

(continues on next page)

(continued from previous page)

```
    res = cudbgAPI->getNextEvent(&event)) {
  switch (event.kind)
      {
      case CUDBG_EVENT_ELF_IMAGE_LOADED:
          //...
          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*.

Enumerations

CUDBGEventKind
Kinds of events sent by the debug engine to the API client.

CUDBGEventQueueType
Application event queue type.

Structs

CUDBGEvent
Event information container.

CUDBGEvent30
Event information container for API version 3.0.

CUDBGEvent32
Event information container for API version 3.2.

CUDBGEvent42
Event information container for API version 4.2.

CUDBGEvent50
Event information container for API version 5.0.

CUDBGEvent55
Event information container for API version 5.5.

CUDBGEventCallbackData
Callback data passed to callback set with setNotifyNewEventCallback function.

CUDBGEventCallbackData40
Callback data passed to callback set with setNotifyNewEventCallback40 function.

CUDBGEventCallbackData41
Callback data passed to callback set with setNotifyNewEventCallback41 function.

Typedefs

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

CUDBGNotifyNewEventCallback31
Function type of the function called to notify debugger of the presence of a new event in the event queue.

CUDBGNotifyNewEventCallback40
Function type of the function called to notify debugger of the presence of a new event in the event queue.

CUDBGNotifyNewEventCallback41
Function type of the function called to notify debugger of the presence of a new event in the event queue.

Variables

CUDBGResult(* *CUDBGAPI_st::acknowledgeEvent30*
Inform the debugger API that synchronous events have been processed.

CUDBGResult(* *CUDBGAPI_st::acknowledgeEvents42*
Inform the debugger API that synchronous events have been processed.

CUDBGResult(* *CUDBGAPI_st::acknowledgeSyncEvents*
Inform the debugger API that synchronous events have been processed.

CUDBGResult(* *CUDBGAPI_st::getErrorStringEx*
Fills a user-provided buffer with an error message encoded as a null-terminated ASCII string.

CUDBGResult(* *CUDBGAPI_st::getNextAsyncEvent50*
Copies the next available event in the asynchronous event queue into 'event' and removes it from the queue.

CUDBGResult(* *CUDBGAPI_st::getNextAsyncEvent55*
Copies the next available event in the asynchronous event queue into 'event' and removes it from the queue.

CUDBGResult(* *CUDBGAPI_st::getNextEvent*
Copies the next available event into 'event' and removes it from the queue.

CUDBGResult(* *CUDBGAPI_st::getNextEvent30*
Copies the next available event in the event queue into 'event' and removes it from the queue.

CUDBGResult(* *CUDBGAPI_st::getNextEvent32*
Copies the next available event in the event queue into 'event' and removes it from the queue.

CUDBGResult(* *CUDBGAPI_st::getNextEvent42*
Copies the next available event in the event queue into 'event' and removes it from the queue.

CUDBGResult(* *CUDBGAPI_st::getNextSyncEvent50*
Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.

CUDBGResult(* *CUDBGAPI_st::getNextSyncEvent55*
Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.

CUDBGResult(* *CUDBGAPI_st::setNotifyNewEventCallback*
Provides the API with the function to call to notify the debugger of a new application or device event.

CUDBGResult(* *CUDBGAPI_st::setNotifyNewEventCallback31*
Provides the API with the function to call to notify the debugger of a new application or device event.

CUDBGResult(* *CUDBGAPI_st::setNotifyNewEventCallback40*
Provides the API with the function to call to notify the debugger of a new application or device event.

CUDBGResult(* *CUDBGAPI_st::setNotifyNewEventCallback41*
Provides the API with the function to call to notify the debugger of a new application or device event.

2.7.1. Enumerations

enum **CUDBGEventKind**

Kinds of events sent by the debug engine to the API client.

Values:

enumerator **CUDBG_EVENT_INVALID** = 0x000

Invalid event.

enumerator **CUDBG_EVENT_ELF_IMAGE_LOADED** = 0x001

The ELF image for a CUDA source module is available.

enumerator **CUDBG_EVENT_KERNEL_READY** = 0x002

A CUDA kernel is about to be launched.

Note: DEPRECATED: This event is unreliable and will be removed in a future release. Since CUDA 13.5, if CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LAZY_FUNCTION_FINALIZATION is enabled, this event will not be sent.

---

enumerator **CUDBG_EVENT_KERNEL_FINISHED** = 0x003

A CUDA kernel has terminated.

Note: DEPRECATED: This event is unreliable and will be removed in a future release. Since CUDA 13.5, if CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LAZY_FUNCTION_FINALIZATION is enabled, this event will not be sent.

---

enumerator **CUDBG_EVENT_INTERNAL_ERROR** = 0x004

An internal error occurred.

The API may be unstable.

enumerator **CUDBG_EVENT_CTX_PUSH** = 0x005

A CUDA context has been pushed.

enumerator **CUDBG_EVENT_CTX_POP** = 0x006

A CUDA context has been popped.

enumerator **CUDBG_EVENT_CTX_CREATE** = 0x007

A CUDA context has been created.

enumerator **CUDBG_EVENT_CTX_DESTROY** = 0x008

A CUDA context has been, popped if pushed, then destroyed.

enumerator **CUDBG_EVENT_TIMEOUT** = 0x009

A timeout event is sent at regular interval.

This event can safely be ignored. Only sent by the classic backend.

enumerator **CUDBG_EVENT_ATTACH_COMPLETE** = 0x00a

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

enumerator **CUDBG_EVENT_DETACH_COMPLETE** = 0x00b

The detach process has completed.

enumerator **CUDBG_EVENT_ELF_IMAGE_UNLOADED** = 0x00c

The ELF image for CUDA kernel(s) no longer available.

enumerator **CUDBG_EVENT_FUNCTIONS_LOADED** = 0x00d

A group of functions/kernels have been loaded Will only be sent if the debug engine capability CUDBG_DEBUGGER_CAPABILITY_LAZY_FUNCTION_LOADING is set.

enumerator **CUDBG_EVENT_ALL_DEVICES_SUSPENDED** = 0x00e

All CUDA devices have been suspended due to a breakpoint hit or an exception.

Does not get sent for GPU events that result in synchronous API method calls, such as singleStepWarp or resumeWarpsUntilPC (unless used in non-blocking mode, in which case it will be sent if the step can't be complete due to a breakpoint or an exception). Will only be sent if the debug engine capability CUDBG_DEBUGGER_CAPABILITY_SUSPEND_EVENTS is set. Note that calling suspendDevice() or suspendAllDevices(false) will not trigger this event even if the device has breakpoints or exceptions to report. Make sure to walk the device state and find the breakpoint/exception status for each warp after calling these APIs.

enumerator **CUDBG_EVENT_CUDA_LOGS_AVAILABLE** = 0x00f

(Async or Sync) CUDA Logs are available for the debugger to consume.

After receiving this event, debuggers should drain all available log entries by repeatedly calling consumeCudaLogs until no more logs are available. For asynchronous CUDA logs, this event is only sent for the first log message that's generated after the client has read all logs

with consumeCudaLogs. For synchronous CUDA logs, this event is sent for each log message that's generated and the emitting thread will wait for the client to acknowledge the event. See *CUDBGAPI_st::setCudaLogRules* for more details. It is not sent by default, and can be enabled via the capability CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LOGS.

enumerator **CUDBG_EVENT_CUDA_LOGS_THRESHOLD_REACHED** = 0x010

(Sync) CUDA Logs buffer has reached an implementation-defined threshold.

The client should call consumeCudaLogs to avoid excessive log buildup. New logs will still be collected even if consumeCudaLogs is not called.

enumerator **CUDBG_EVENT_SINGLE_STEP_COMPLETE** = 0x011

(Sync) Asynchronous single step operation has been completed.

Call singleStepWarp or resumeWarpsUntilPC with CUDBG_SINGLE_STEP_FLAGS_NON_BLOCKING to enable this event. This event is only sent if the single step operation was completed. If it's interrupted by a breakpoint or an exception, a CUDBG_EVENT_ALL_DEVICES_SUSPENDED event will be sent instead.

enumerator **CUDBG_EVENT_CUDA_LOGS_RULESET_CHANGED** = 0x012

(Async) The CUDA log ruleset has changed.

This event is sent after the new ruleset has been applied. The client can call consumeCudaLogs to drain the log buffer, after which it's guaranteed that the new logs will only match the ruleset with the index larger than or equal to the ruleset index of this event. This allows the clients to stop tracking the old rulesets.

enum **CUDBGEventQueueType**

Application event queue type.

Values:

enumerator **CUDBG_EVENT_QUEUE_TYPE_SYNC** = 0

Synchronous event queue.

enumerator **CUDBG_EVENT_QUEUE_TYPE_ASYNC** = 1

Asynchronous event queue.

2.7.2. 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.

Param data
- pointer to the event callback data.

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.

Note: DEPRECATED: Use CUDBGNotifyNewEventCallback instead.

Param data
- pointer to the event callback data.

typedef void (*__CUDBGNotifyNewEventCallback40__)(*CUDBGEventCallbackData40* *data)

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

Note: DEPRECATED: Use CUDBGNotifyNewEventCallback instead.

Param data
- pointer to the event callback data.

typedef void (*__CUDBGNotifyNewEventCallback41__)(*CUDBGEventCallbackData41* *data)

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

Note: DEPRECATED: Use CUDBGNotifyNewEventCallback instead.

Param data
- pointer to the event callback data.

2.8. General

Macros

CUDBG_APICLIENT_PID
Name of the global variable containing the API client PID.

CUDBG_APICLIENT_REVISION
Name of the global variable containing the API client revision.

CUDBG_API_VERSION_MAJOR
Major release version number.

CUDBG_API_VERSION_MINOR
Minor release version number.

CUDBG_API_VERSION_REVISION
API revision number.

CUDBG_ATTACH_HANDLER_AVAILABLE
Name of the global variable containing the attach handler available flag.

CUDBG_DEBUGGER_CAPABILITIES
Name of the global variable containing the debug engine capabilities bitmask.

CUDBG_DEBUGGER_INITIALIZED
Name of the global variable containing the debug engine initialized flag.

CUDBG_ENABLE_LAUNCH_BLOCKING
Name of the global variable containing the enable launch blocking flag.

CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD
Name of the global variable containing the debug engine attach procedure initiation FD.

CUDBG_IPC_FLAG_NAME
Name of the global variable containing the IPC flag.

CUDBG_MAX_DEVICES
Maximum number of supported devices.

CUDBG_MAX_LANES
Maximum number of lanes per warp.

CUDBG_MAX_LOG_LEN
Maximum length of a single CUDA log message.

CUDBG_MAX_SMS
Maximum number of SMs per device.

CUDBG_MAX_WARPS
Maximum number of warps per SM.

CUDBG_MAX_WARP_BARRIERS
Maximum number of convergence barriers per warp.

CUDBG_PRE_INIT
Name of the pre-init global function.

CUDBG_REPORTED_DRIVER_API_ERROR_CODE
Name of the global variable containing the code of the driver API error being reported.

CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_ADDR
Name of the global variable containing the address of the name of the function in which the driver API error occurred.

CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_SIZE
Name of the global variable containing the size of the name of the function in which the driver API error occurred.

CUDBG_REPORTED_DRIVER_API_ERROR_NAME_ADDR
Name of the global variable containing the address where the driver API error name is stored.

CUDBG_REPORTED_DRIVER_API_ERROR_NAME_SIZE
Name of the global variable containing the size of the driver API error name being reported.

CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE
Name of the global variable containing the driver API error source.

CUDBG_REPORTED_DRIVER_API_ERROR_STRING_ADDR
Name of the global variable containing the address where the driver API error string is stored.

CUDBG_REPORTED_DRIVER_API_ERROR_STRING_SIZE
Name of the global variable containing the size of the driver API error string being reported.

CUDBG_REPORTED_DRIVER_INTERNAL_ERROR_CODE
Name of the global variable containing the driver internal error code.

CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED
Name of the global function that's called to report the completion of the attach procedure.

CUDBG_REPORT_DRIVER_API_ERROR
Name of the global function that's called to report a driver API error.

CUDBG_REPORT_DRIVER_API_ERROR_FLAGS
Name of the global variable containing the driver API error flag.

CUDBG_REPORT_DRIVER_INTERNAL_ERROR
Name of the global function that's called to report an internal driver error.

CUDBG_RESUME_FOR_ATTACH_DETACH
Name of the global variable containing the resume for attach detach flag.

CUDBG_RPC_ENABLED
Name of the global variable containing the RPC enabled flag.

CUDBG_SESSION_ID
Name of the global variable containing the session ID.

CUDBG_USE_EXTERNAL_DEBUGGER
Name of the global variable containing the external debug engine in use flag.

Enumerations

CUDBGCapabilityFlags
Debug engine capability flags.

CUDBGReportDriverApiErrorFlags
API error reporting flags.

CUDBGReportedDriverApiErrorSource
Driver API error source.

CUDBGResult
Result values of all the API routines.

Functions

void *cudbgApiAttach*(**void**)
Attach to a CUDA application.

void *cudbgApiDetach*(**void**)
Detach from a CUDA application.

void *cudbgApiInit*(**uint32_t arg**)
Initialize the CUDA Debugger API.

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.

const char * *cudbgGetErrorString*(**CUDBGResult error**)
Returns the string representation of a result value.

void *cudbgPreInit*()
Empty global function that gets called before CUDA driver initialization.

void *cudbgReportAttachProcedureFinished*(**void**)
Report that the attach procedure has finished.

void ***cudbgReportDriverApiError*****(void)**
Report a driver API error.

void ***cudbgReportDriverInternalError*****(void)**
Report a driver internal error.

Structs

CuDim2
2-dimensional coordinates for threads, blocks, etc.

CuDim3
3-dimensional coordinates for threads, blocks, etc.

Typedefs

CUDBGAPI
CUDA Debugger API methods.

2.8.1. Macros

CUDBG_APICLIENT_PID cudbgApiClientPid

Name of the global variable containing the API client PID.

This variable is set by the API client to identify itself.

CUDBG_APICLIENT_REVISION cudbgApiClientRevision

Name of the global variable containing the API client revision.

This variable is set by the API client to identify the API revision it wants to use.

CUDBG_API_VERSION_MAJOR 13

Major release version number.

This matches the major version of the CUDA driver exposing this API.

CUDBG_API_VERSION_MINOR 5

Minor release version number.

This matches the minor version of the CUDA driver exposing this API.

CUDBG_API_VERSION_REVISION 200

API revision number.

This number is incremented every time changes are made to the API.

CUDBG_ATTACH_HANDLER_AVAILABLE cudbgAttachHandlerAvailable

Name of the global variable containing the attach handler available flag.

This variable is set by the debug engine to indicate that the attach handler is available.

CUDBG_DEBUGGER_CAPABILITIES cudbgDebuggerCapabilities

Name of the global variable containing the debug engine capabilities bitmask.

This variable is set by the API client to indicate the requested capabilities. See the CUDBGCapabilityFlags enum for the list of available capabilities.

CUDBG_DEBUGGER_INITIALIZED cudbgDebuggerInitialized

Name of the global variable containing the debug engine initialized flag.

This variable is set by the debug engine to indicate that it has been initialized.

CUDBG_ENABLE_LAUNCH_BLOCKING cudbgEnableLaunchBlocking

Name of the global variable containing the enable launch blocking flag.

This variable is set by the API client to enable blocking launches of CUDA kernels.

CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD cudbgInitiateDebuggerAttachProcedureFd

Name of the global variable containing the debug engine attach procedure initiation FD.

If this FD is not -1, write a single byte (any value) to it to initiate the CUDA debug engine attach procedure. After the request was received and the attach procedure has finished, the CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED function is called.

CUDBG_IPC_FLAG_NAME cudbgIpcFlag

Name of the global variable containing the IPC flag.

This variable is set by the API client to indicate that it is ready to attach to a running CUDA application.

CUDBG_MAX_DEVICES 64

Maximum number of supported devices.

CUDBG_MAX_LANES 32

Maximum number of lanes per warp.

CUDBG_MAX_LOG_LEN 256

Maximum length of a single CUDA log message.

CUDBG_MAX_SMS 256

Maximum number of SMs per device.

CUDBG_MAX_WARPS 64

Maximum number of warps per SM.

CUDBG_MAX_WARP_BARRIERS 16

Maximum number of convergence barriers per warp.

CUDBG_PRE_INIT *cudbgPreInit*

Name of the pre-init global function.

The API client can set a breakpoint on this function to be notified before the CUDA driver starts its initialization.

CUDBG_REPORTED_DRIVER_API_ERROR_CODE cudbgReportedDriverApiErrorCode

Name of the global variable containing the code of the driver API error being reported.

This variable is set by the debug engine when a driver API error is reported.

CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_ADDR
cudbgReportedDriverApiErrorFuncNameAddr

Name of the global variable containing the address of the name of the function in which the driver API error occurred.

This variable is set by the debug engine when a driver API error is reported.

CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_SIZE
cudbgReportedDriverApiErrorFuncNameSize

Name of the global variable containing the size of the name of the function in which the driver API error occurred.

This variable is set by the debug engine when a driver API error is reported.

CUDBG_REPORTED_DRIVER_API_ERROR_NAME_ADDR cudbgReportedDriverApiErrorNameAddr

Name of the global variable containing the address where the driver API error name is stored.

This variable is set by the debug engine when a driver API error is reported.

CUDBG_REPORTED_DRIVER_API_ERROR_NAME_SIZE cudbgReportedDriverApiErrorNameSize

Name of the global variable containing the size of the driver API error name being reported.

This variable is set by the debug engine when a driver API error is reported.

CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE cudbgReportedDriverApiErrorSource

Name of the global variable containing the driver API error source.

This variable is set by the debug engine when a driver API error is reported. See the CUDBGReportedDriverApiErrorSource enum for the list of available sources.

CUDBG_REPORTED_DRIVER_API_ERROR_STRING_ADDR cudbgReportedDriverApiErrorStringAddr

Name of the global variable containing the address where the driver API error string is stored.

This variable is set by the debug engine when a driver API error is reported.

CUDBG_REPORTED_DRIVER_API_ERROR_STRING_SIZE cudbgReportedDriverApiErrorStringSize

Name of the global variable containing the size of the driver API error string being reported.

This variable is set by the debug engine when a driver API error is reported.

CUDBG_REPORTED_DRIVER_INTERNAL_ERROR_CODE cudbgReportedDriverInternalErrorCode

Name of the global variable containing the driver internal error code.

This variable is set by the debug engine when an internal driver error is reported.

CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED *cudbgReportAttachProcedureFinished*

Name of the global function that's called to report the completion of the attach procedure.

The API client can set a breakpoint on this function to be notified about the completion of the attach procedure. This function is called after the attach is done after signalling CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD.

CUDBG_REPORT_DRIVER_API_ERROR *cudbgReportDriverApiError*

Name of the global function that's called to report a driver API error.

The API client can set a breakpoint on this function to be notified about driver API errors.

CUDBG_REPORT_DRIVER_API_ERROR_FLAGS cudbgReportDriverApiErrorFlags

Name of the global variable containing the driver API error flag.

This variable is set by the API client to indicate which driver API errors should be reported. See the CUDBGReportDriverApiErrorFlags enum for the list of available flags.

CUDBG_REPORT_DRIVER_INTERNAL_ERROR *cudbgReportDriverInternalError*

Name of the global function that's called to report an internal driver error.

The API client can set a breakpoint on this function to be notified about internal driver errors.

CUDBG_RESUME_FOR_ATTACH_DETACH cudbgResumeForAttachDetach

Name of the global variable containing the resume for attach detach flag.

This variable is set by the debug engine to indicate that the API client should resume the CUDA application (including the CPU) to complete the attach or detach procedure.

CUDBG_RPC_ENABLED cudbgRpcEnabled

Name of the global variable containing the RPC enabled flag.

This variable is only used by debuggers that use the RPCD interface (e.g. CUDA-GDB).

CUDBG_SESSION_ID cudbgSessionId

Name of the global variable containing the session ID.

This variable is set by the API client to identify the session.

CUDBG_USE_EXTERNAL_DEBUGGER cudbgUseExternalDebugger

Name of the global variable containing the external debug engine in use flag.

Can be read to detect whether the external debug engine implementation (libcudadebugger.so) is used or not.

Note: Since CUDA 13.1, the external debug engine implementation is always used.

2.8.2. Enumerations

enum **CUDBGCapabilityFlags**

Debug engine capability flags.

Clients should request the capabilities they want by setting the CUDBG_DEBUGGER_CAPABILITIES global variable and then checking the supported capabilities by calling the getSupportedDebuggerCapabilities() API method and adjusting their behavior accordingly. Capabilities requested but not supported by the debug engine will be ignored and should not be relied upon.

Values:

enumerator **CUDBG_DEBUGGER_CAPABILITY_NONE** = 0

No capabilities.

enumerator **CUDBG_DEBUGGER_CAPABILITY_LAZY_FUNCTION_LOADING** = (1 << 0)

Lazy function loading.

Static flag: cannot be changed after initialization. Requesting this capability will enable CUDBG_EVENT_FUNCTIONS_LOADED events to be sent. This capability should not be requested until the API client is prepared to handle these events.

enumerator **CUDBG_DEBUGGER_CAPABILITY_SUSPEND_EVENTS** = (1 << 1)

Suspend events.

Static flag: cannot be changed after initialization. Requesting this capability will enable CUDBG_EVENT_ALL_DEVICES_SUSPENDED events to be sent. This capability should not be requested until the API client is prepared to handle these events.

enumerator **CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS** = (1 << 2)

Report exceptions in exited warps.

Static flag: cannot be changed after initialization. Requesting this capability will enable reporting of exceptions in exited warps. This capability should not be requested until the API client is prepared to handle such situations.

enumerator **CUDBG_DEBUGGER_CAPABILITY_NO_CONTEXT_PUSH_POP_EVENTS** = (1 << 3)

No context push/pop events.

Static flag: cannot be changed after initialization. Requesting this capability will disable the CUDBG_EVENT_CONTEXT_PUSH and CUDBG_EVENT_CONTEXT_POP events. This capability should be requested if the push/pop events are not used by the API client.

enumerator **CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LOGS** = (1 << 4)

Enable CUDA logs.

Dynamic flag: can be changed after initialization. Requesting this capability will enable CUDA log capture by the debug engine and cause CUDBG_EVENT_CUDA_LOGS_AVAILABLE and CUDBG_EVENT_CUDA_LOGS_THRESHOLD_REACHED events to be sent. This capability should not be requested until the API client is prepared to handle these events.

enumerator **CUDBG_DEBUGGER_CAPABILITY_COLLECT_CPU_CALL_STACK_FOR_KERNEL_LAUNCHES** = (1 << 5)

Collect CPU call stack for kernel launches.

Dynamic flag: can be changed after initialization. Requesting this capability will enable collection of CPU call stack for kernel launches. This capability should not be requested if the API client does not plan on calling the readCPUCallStack() API.

enumerator **CUDBG_DEBUGGER_CAPABILITY_FLUSH_PRINTF_ON_SUSPEND** = (1 << 6)

Flush printf on suspend.

Static flag: cannot be changed after initialization. Requesting this capability will enable flushing of the CUDA printf output on suspend. This capability should generally be requested.

enumerator **CUDBG_DEBUGGER_CAPABILITY_BREAK_ON_LAUNCH** = (1 << 7)

Enable break on launch feature.

Static flag: cannot be changed after initialization. Requesting this capability will enable the break on launch feature. It may impact launch performance for light workloads. This capability should be requested if break on launch is used.

enumerator **CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LAZY_FUNCTION_FINALIZATION** = (1 << 8)

Enable CUDA Lazy Function Finalization.

Static flag: cannot be changed after initialization. Requesting this capability will enable the CUDA Lazy Function Finalization feature. This capability requires the API client to be prepared to handle fragment load events. This capability prevents the use of getElfImage calls. Enabling this capability without enabling CUDBG_DEBUGGER_CAPABILITY_LAZY_FUNCTION_LOADING is not allowed and will result in an error. This capability will prevent deprecated events CUDBG_EVENT_KERNEL_READY and CUDBG_EVENT_KERNEL_FINISHED from being sent.

enum **CUDBGReportDriverApiErrorFlags**

API error reporting flags.

Values:

enumerator **CUDBG_REPORT_DRIVER_API_ERROR_FLAGS_NONE** = 0x0000

No flags set (default behavior).

enumerator **CUDBG_REPORT_DRIVER_API_ERROR_FLAGS_SUPPRESS_NOT_READY** = (1U << 0)

When set, cudaErrorNotReady/cuErrorNotReady will not be reported.

enum **CUDBGReportedDriverApiErrorSource**

Driver API error source.

Values:

enumerator **CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE_NONE** = 0x000
No error/source.

enumerator **CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE_DRIVER** = 0x001
The error originates from the CUDA Driver API.

enumerator **CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE_RUNTIME** = 0x002
The error originates from the CUDA Runtime API.

enum **CUDBGResult**
Result values of all the API routines.
Values:

enumerator **CUDBG_SUCCESS** = 0x0000
The API call executed successfully.

enumerator **CUDBG_ERROR_UNKNOWN** = 0x0001
Error type not listed below.

enumerator **CUDBG_ERROR_BUFFER_TOO_SMALL** = 0x0002
Cannot copy all the queried data into the buffer argument.

enumerator **CUDBG_ERROR_UNKNOWN_FUNCTION** = 0x0003
Function cannot be found in the CUDA kernel.

enumerator **CUDBG_ERROR_INVALID_ARGS** = 0x0004
Wrong use of arguments (NULL pointer, illegal value,....).

enumerator **CUDBG_ERROR_UNINITIALIZED** = 0x0005
The API has not yet been properly initialized.

enumerator **CUDBG_ERROR_INVALID_COORDINATES** = 0x0006
Invalid block or thread coordinates were provided.

enumerator **CUDBG_ERROR_INVALID_MEMORY_SEGMENT** = 0x0007
Invalid memory segment requested.

enumerator **CUDBG_ERROR_INVALID_MEMORY_ACCESS** = 0x0008
Requested address (+size) is not within proper segment boundaries.

enumerator **CUDBG_ERROR_MEMORY_MAPPING_FAILED** = 0x0009
Memory is not mapped and cannot be mapped.

enumerator **CUDBG_ERROR_INTERNAL** = 0x000a
A debug engine internal error occurred.

enumerator **CUDBG_ERROR_INVALID_DEVICE** = 0x000b
Specified device cannot be found.

enumerator **CUDBG_ERROR_INVALID_SM** = 0x000c
Specified sm cannot be found.

enumerator **CUDBG_ERROR_INVALID_WARP** = 0x000d
Specified warp cannot be found.

enumerator **CUDBG_ERROR_INVALID_LANE** = 0x000e
Specified lane cannot be found.

enumerator **CUDBG_ERROR_SUSPENDED_DEVICE** = 0x000f
The requested operation is not allowed when the device is suspended.

enumerator **CUDBG_ERROR_RUNNING_DEVICE** = 0x0010
Device is running and not suspended.

enumerator **CUDBG_ERROR_RESERVED_0** = 0x0011
Reserved error code.

enumerator **CUDBG_ERROR_INVALID_ADDRESS** = 0x0012
Address is out-of-range.

enumerator **CUDBG_ERROR_INCOMPATIBLE_API** = 0x0013
The requested API is not available.

enumerator **CUDBG_ERROR_INITIALIZATION_FAILURE** = 0x0014
The API could not be initialized.

enumerator **CUDBG_ERROR_INVALID_GRID** = 0x0015
The specified grid is not valid.

enumerator **CUDBG_ERROR_NO_EVENT_AVAILABLE** = 0x0016
The event queue is empty and there is no event left to be processed.

enumerator **CUDBG_ERROR_SOME_DEVICES_WATCHDOGGED** = 0x0017
Some devices were excluded because they have a watchdog associated with them.

enumerator **CUDBG_ERROR_ALL_DEVICES_WATCHDOGGED** = 0x0018
All devices were exclude because they have a watchdog associated with them.

enumerator **CUDBG_ERROR_INVALID_ATTRIBUTE** = 0x0019
Specified attribute does not exist or is incorrect.

enumerator **CUDBG_ERROR_ZERO_CALL_DEPTH** = 0x001a

No function calls have been made on the device.

enumerator **CUDBG_ERROR_INVALID_CALL_LEVEL** = 0x001b

Specified call level is invalid.

enumerator **CUDBG_ERROR_COMMUNICATION_FAILURE** = 0x001c

Communication error between the debug engine and the application.

enumerator **CUDBG_ERROR_INVALID_CONTEXT** = 0x001d

Specified context cannot be found.

enumerator **CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM** = 0x001e

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

enumerator **CUDBG_ERROR_MEMORY_UNMAPPING_FAILED** = 0x001f

Requested address is not mapped and cannot be unmapped.

enumerator **CUDBG_ERROR_INCOMPATIBLE_DISPLAY_DRIVER** = 0x0020

The display driver is incompatible with the API.

enumerator **CUDBG_ERROR_INVALID_MODULE** = 0x0021

The specified module is not valid.

enumerator **CUDBG_ERROR_LANE_NOT_IN_SYSCALL** = 0x0022

The specified lane is not inside a device syscall.

enumerator **CUDBG_ERROR_RESERVED_1** = 0x0023

Reserved error code.

enumerator **CUDBG_ERROR_INVALID_ENVVAR_ARGS** = 0x0024

Some environment variable's value is invalid.

enumerator **CUDBG_ERROR_OS_RESOURCES** = 0x0025

Error while allocating resources from the OS.

enumerator **CUDBG_ERROR_FORK_FAILED** = 0x0026

Error while forking the debug engine process.

enumerator **CUDBG_ERROR_NO_DEVICE_AVAILABLE** = 0x0027

No CUDA capable device was found.

enumerator **CUDBG_ERROR_ATTACH_NOT_POSSIBLE** = 0x0028

Attaching to the CUDA program is not possible.

enumerator **CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE** = 0x0029

The resumeWarpsUntilPC() API is not possible, use resumeDevice() or singleStepWarp() instead.

enumerator **CUDBG_ERROR_INVALID_WARP_MASK** = 0x002a

Specified warp mask is zero, or contains invalid warps.

enumerator **CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS** = 0x002b

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

enumerator **CUDBG_ERROR_RECURSIVE_API_CALL** = 0x002c

Debug API entry point called from within a debug API callback.

enumerator **CUDBG_ERROR_MISSING_DATA** = 0x002d

The requested data is missing.

enumerator **CUDBG_ERROR_NOT_SUPPORTED** = 0x002e

Attempted operation is not supported.

enumerator **CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT** = 0x002f

The current breakpoint state conflicts with the requested operation.

2.8.3. Functions

void **cudbgApiAttach**(void)

Attach to a CUDA application.

Remotely called by the API client during the attach procedure to attach to the CUDA application.

void **cudbgApiDetach**(void)

Detach from a CUDA application.

Remotely called by the API client during the detach procedure to detach from the CUDA application.

void **cudbgApiInit**(uint32_t arg)

Initialize the CUDA Debugger API.

Remotely called by the API client during the attach procedure to initialize the debugger API.

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

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

See also:

cudbgGetAPIVersion

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,

Returns
CUDBG_SUCCESS,

Returns
CUDBG_ERROR_INCOMPATIBLE_API

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

Get the API version supported by the CUDA driver.

See also:

cudbgGetAPI

Parameters

- **major** – - the major version number
- **minor** – - the minor version number
- **rev** – - the revision version number

Returns
CUDBG_ERROR_INVALID_ARGS,

Returns
CUDBG_SUCCESS

static inline const char ***cudbgGetErrorString**(*CUDBGResult* error)

Returns the string representation of a result value.

It is preferred to use this function instead of indexing CUDBGResultNames directly as future versions of the old API methods might start returning new result values.

Parameters
error – The result value to get the string representation of.

Returns
The string representation of the result value or "*UNDEFINED*" if the result value is not recognized.

void **cudbgPreInit**()

Empty global function that gets called before CUDA driver initialization.

The API client can set a breakpoint on this function to be notified before the CUDA driver starts its initialization.

void **cudbgReportAttachProcedureFinished**(void)
Report that the attach procedure has finished.

The API client can set a breakpoint on this function to be notified that the attach procedure has finished.

void **cudbgReportDriverApiError**(void)
Report a driver API error.

The API client can set a breakpoint on this function to be notified about driver API errors.

void **cudbgReportDriverInternalError**(void)
Report a driver internal error.

The API client can set a breakpoint on this function to be notified about driver internal errors.

2.8.4. Typedefs

typedef const struct *CUDBGAPI_st* ***CUDBGAPI**
CUDA Debugger API methods.

2.9. Grid Properties

Enumerations

CUDBGAttribute
Queryable grid attributes.

CUDBGGridStatus
Grid status.

CUDBGKernelLaunchNotifyMode
Kernel launch notification mode.

CUDBGKernelOrigin
Kernel origin.

CUDBGKernelType
Kernel types.

Structs

CUDBGAttributeValuePair
Grid attribute-value pair.

CUDBGGridInfo
Information about a CUDA grid.

CUDBGGridInfo120
Grid info.

CUDBGGridInfo55
Grid info.

Variables

CUDBGResult(* *CUDBGAPI_st::getBlockDim*
Get the dimensions of the given block.

CUDBGResult(* *CUDBGAPI_st::getClusterDim*
Get the number of blocks in the given cluster.

CUDBGResult(* *CUDBGAPI_st::getClusterDim120*
Get the number of blocks in the given cluster.

CUDBGResult(* *CUDBGAPI_st::getElfImage*
Get the relocated or non-relocated ELF image and size for the grid on the given device.

CUDBGResult(* *CUDBGAPI_st::getElfImage32*
Get the relocated or non-relocated ELF image and size for the grid on the given device.

CUDBGResult(* *CUDBGAPI_st::getGridAttribute*
Get the value of a grid attribute.

CUDBGResult(* *CUDBGAPI_st::getGridAttributes*
Get several grid attribute values in a single API call.

CUDBGResult(* *CUDBGAPI_st::getGridDim*
Get the dimensions in blocks of the given grid.

CUDBGResult(* *CUDBGAPI_st::getGridDim32*
Get the dimensions of the given grid.

CUDBGResult(* *CUDBGAPI_st::getGridInfo*
Get information about the specified grid.

CUDBGResult(* *CUDBGAPI_st::getGridInfo120*
Get information about the specified grid.

CUDBGResult(* *CUDBGAPI_st::getGridInfo55*
Get information about the specified grid.

CUDBGResult(* *CUDBGAPI_st::getGridStatus*
Check whether the grid corresponding to the ID is still present on the device.

CUDBGResult(* *CUDBGAPI_st::getGridStatus50*
Check whether the grid corresponding to the ID is still present on the device.

CUDBGResult(* *CUDBGAPI_st::getTID*
Get the ID of the Linux thread hosting the CUDA context active at the given coordinates.

2.9.1. Enumerations

enum **CUDBGAttribute**

Queryable grid attributes.

Values:

enumerator **CUDBG_ATTR_GRID_LAUNCH_BLOCKING** = 0x000

Whether the launch is synchronous (blocking) or not.

enumerator **CUDBG_ATTR_GRID_TID** = 0x001
The id of the host thread that launched the grid.

enum **CUDBGGridStatus**
Grid status.
Values:

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

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

enumerator **CUDBG_GRID_STATUS_ACTIVE**
The grid is currently running on the HW.

enumerator **CUDBG_GRID_STATUS_SLEEPING**
The grid is on the device, doing a join.

enumerator **CUDBG_GRID_STATUS_TERMINATED**
The grid has finished executing.

enumerator **CUDBG_GRID_STATUS_UNDETERMINED**
The grid is either QUEUED or TERMINATED.

enum **CUDBGKernelLaunchNotifyMode**
Kernel launch notification mode.
Values:

enumerator **CUDBG_KNL_LAUNCH_NOTIFY_EVENT** = 0x000
Kernel launches generate launch notification events.

enumerator **CUDBG_KNL_LAUNCH_NOTIFY_DEFER** = 0x001
Kernel launches do not generate any notification.

enum **CUDBGKernelOrigin**
Kernel origin.
Values:

enumerator **CUDBG_KNL_ORIGIN_CPU** = 0x000
The kernel was launched from the CPU.

enumerator **CUDBG_KNL_ORIGIN_GPU** = 0x001
The kernel was launched from the GPU.

enum **CUDBGKernelType**

Kernel types.

Values:

enumerator **CUDBG_KNL_TYPE_UNKNOWN** = 0x000

Unknown kernel type.

Fall-back value.

enumerator **CUDBG_KNL_TYPE_SYSTEM** = 0x001

System kernel, launched by the CUDA driver (cudaMemset, ...).

enumerator **CUDBG_KNL_TYPE_APPLICATION** = 0x002

Application kernel, launched by the application (user-defined or libraries).

2.10. Initialization

Variables

CUDBGResult(* *CUDBGAPI_st::clearAttachState*
Clear attach-specific state prior to detach.

CUDBGResult(* *CUDBGAPI_st::finalize*
Finalize the API, shutting down the debugging session.

CUDBGResult(* *CUDBGAPI_st::getSupportedDebuggerCapabilities*
Returns debug agent capabilities that are supported by this version of the API.

CUDBGResult(* *CUDBGAPI_st::initialize*
Initialize the API.

CUDBGResult(* *CUDBGAPI_st::initializeAttachStub*
Initialize the attach stub.

CUDBGResult(* *CUDBGAPI_st::requestCleanupOnDetach*
Request for cleanup of driver state when detaching.

CUDBGResult(* *CUDBGAPI_st::requestCleanupOnDetach55*
Request for cleanup of driver state when detaching.

Breakpoints

DWARF Utilities

Device Execution Control

Device Properties

Device State Alteration

Device State Inspection

Events

One of those events will create a *CUDBGEvent* :

General

Grid Properties

Initialization

Chapter 3. Structs

3.1. CUDBGAPI_st

struct **CUDBGAPI_st**

CUDA debugger API methods.

Public Members

CUDBGResult (***acknowledgeEvent30**)(*CUDBGEvent30* *event)

Inform the debugger API that synchronous events have been processed.

Behaves exactly like acknowledgeSyncEvents (the event parameter is ignored).

Since CUDA 3.0.

See also:

acknowledgeSyncEvents

Note: DEPRECATED in CUDA 3.1: Use *acknowledgeSyncEvents* instead.

Param event
[in] - pointer to the event that has been processed

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE

CUDBGResult (***acknowledgeEvents42**)(void)

Inform the debugger API that synchronous events have been processed.

Behaves exactly like acknowledgeSyncEvents.

Since CUDA 3.1.

See also:

acknowledgeSyncEvents

Note: DEPRECATED in CUDA 5.0: Use *acknowledgeSyncEvents* instead.

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE

CUDBGResult (***acknowledgeSyncEvents**)(void)

Inform the debugger API that synchronous events have been processed.

This resumes any process that was interrupted by the synchronous event (e.g. a context creation, a module load, etc.). This method always acknowledges only those SYNC events that have been read with getNextEvent (or its deprecated variants). SYNC events that haven't been read are not acknowledged and will continue to prevent their corresponding processes from proceeding. ASYNC events do not require acknowledgement.

Since CUDA 5.0.

See also:

getNextEvent

See also:

setNotifyNewEventCallback

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE

CUDBGResult (***clearAttachState***)(void)

Clear attach-specific state prior to detach.

This call prepares the API for detaching. See the “Attaching and Detaching” section for more information.

Since CUDA 5.0.

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***consumeCudaLogs***)(*CUDBGCudaLogMessage* *logMessages, uint32_t numMessages, uint32_t *numConsumed)

Get CUDA error log entries.

This consumes the log entries, so they will not be available in subsequent calls. This functionality is only available if the CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LOGS capability is enabled.

This function can be called from the notification callback function.

Since CUDA 13.4.

See also:

setCudaLogRules

Param logMessages
[out] - client-allocated array to store log entries

Param numMessages
[in] - capacity of the logMessages array, in number of elements

Param numConsumed
[out] - returned number of entries written to logMessages

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***consumeCudaLogs129***)(*CUDBGCudaLogMessage129* *logMessages, uint32_t numMessages, uint32_t *numConsumed)

Get CUDA error log entries.

This consumes the log entries, so they will not be available in subsequent calls. This functionality is only available if the CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LOGS capability is enabled.

Since CUDA 12.9.

See also:

consumeCudaLogs

Note: Since CUDA 13.4, this function can be called from the notification callback function.

Note: DEPRECATED in CUDA 13.4: Use *consumeCudaLogs* instead.

Param logMessages
[out] - client-allocated array to store log entries

Param numMessages
[in] - capacity of the logMessages array, in number of elements

Param numConsumed
[out] - returned number of entries written to logMessages

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***disableBreakpoint***)(*CUDBGBreakpointHandle* handle)

Disable a breakpoint specified by its handle.

Disabling/enabling a breakpoint might be faster than removing and inserting it again.

Since CUDA 13.2.

See also:

enableBreakpoint

See also:

getWarpHitBreakpoint

See also:

insertBreakpoint

See also:

isBreakpointEnabled

See also:

removeBreakpoint

Param handle
[in] - the breakpoint handle. If it's *CUDBG_BREAKPOINT_HANDLE_ALL_USER_BREAKPOINTS*, disable all breakpoints inserted by the client (except the break-on-launch breakpoint). It's fine to use ALL_USER_BREAKPOINTS when there are no breakpoints added.

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

CUDBGResult (***disassemble**)(uint32_t dev, uint64_t addr, uint32_t *instSize, char *buf, uint32_t sz)

Disassemble instruction at instruction address.

This method does not guarantee any specific output format and its result should be treated as plain text.

Since CUDA 3.0.

Param dev
[in] - device index

Param addr
[in] - instruction address

Param instSize
[out] - instruction size (32 or 64 bits)

Param buf
[out] - disassembled instruction buffer

Param sz
[in] - disassembled instruction buffer size

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`enableBreakpoint`)(*CUDBGBreakpointHandle* handle)

Enable a breakpoint specified by its handle.

Disabling/enabling a breakpoint might be faster than removing and inserting it again.

Since CUDA 13.2.

See also:

disableBreakpoint

See also:

getWarpHitBreakpoint

See also:

insertBreakpoint

See also:

isBreakpointEnabled

See also:

removeBreakpoint

Param handle
[in] - the breakpoint handle. If it's *CUDBG_BREAKPOINT_HANDLE_ALL_USER_BREAKPOINTS*, enable all breakpoints inserted by the client (except the break-on-launch breakpoint). It's fine to use ALL_USER_BREAKPOINTS when there are no breakpoints added.

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

CUDBGResult (***executeInternalCommand**)(const char *command, char *resultBuffer, uint32_t sizeInBytes)

Execute an internal command (not available in public driver builds)

Always returns CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 12.6.

Param command
[in] - the command name and arguments

Param resultBuffer
[out] - the destination buffer

Param sizeInBytes
[in] - buffer size in bytes

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***finalize**)(void)

Finalize the API, shutting down the debugging session.

Since CUDA 3.0.

See also:

initialize

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***generateCoredump***)(const char *filename, *CUDBGCoredumpGenerationFlags* flags)

Generate a coredump for the current GPU state.

Since CUDA 12.3.

Param filename
[in] - target coredump file name

Param flags
[in] - coredump generation flags/options

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getAdjustedCodeAddress***)(uint32_t dev, uint64_t address, uint64_t *adjustedAddress, *CUDBGAdjAddrAction* adjAction)

Get the adjusted code address for a given code address for a given device.

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

Since CUDA 5.5.

See also:

setBreakpoint

Param dev
[in] - device index

Param addr
[in] - instruction address

Param adjustedAddress
[out] - adjusted address

Param adjAction
[in] - whether the adjusted next, previous or current address is needed

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getBindlessConstAddress**)(uint32_t dev, uint64_t header, uint64_t *address, uint32_t *size)

Get bindless constant GPU VA and size from its header.

Since CUDA 13.4.

This function does not validate the provided header. If the header is invalid, the function would still return success, but the returned address and size would be invalid.

See also:

getConstBankAddress

Param dev
[in] - device index

Param header
[in] - bindless constant header

Param address
[out] - GPU VA of the start of the mapped bindless constant memory region

Param size
[out] - size of the mapped bindless constant memory region

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getBlockDim**)(uint32_t dev, uint32_t sm, uint32_t wp, *CuDim3* *blockDim)

Get the dimensions of the given block.

Since CUDA 3.0.

See also:

getClusterDim

See also:

getGridDim

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param blockDim
[out] - the returned number of threads in the block

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getCbuWarpState***)(uint32_t dev, uint32_t sm, uint64_t warpMask, *CUDBGCbuWarpState* *warpStates, uint32_t numWarpStates)

Get the CBU state of a given warp.

Since CUDA 12.9.

Param dev
[in] - device index

Param sm
[in] - SM index

Param warpMask
[in] - bitmask of the warps which states should be returned in warpStates

Param warpStates
[out] - pointer to the array of *CUDBGCbuWarpState* structures

Param numWarpStates
[in] - number of elements in warpStates array

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getClusterDim***)(uint32_t dev, uint32_t sm, uint32_t wp, *CuDim3* *clusterDim)

Get the number of blocks in the given cluster.

Since CUDA 12.7.

See also:

getBlockDim

See also:

getGridDim

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param clusterDim
[out] - the returned number of blocks in the cluster

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`getClusterDim120`)(uint32_t dev, uint64_t gridId64, *CuDim3* *clusterDim)

Get the number of blocks in the given cluster.

Behaves like getClusterDim, but takes a grid ID instead of warp coordinates. In newer GPU architectures, it's possible to have different warps belong to blocks of clusters of different size.

Since CUDA 12.0.

See also:

getClusterDim

Note: DEPRECATED in CUDA 12.7: Use *getClusterDim* instead.

Param dev
[in] - device index

Param gridId64
[in] - grid ID

Param clusterDim
[out] - the returned number of blocks in the cluster

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getClusterExceptionTargetBlock**)(uint32_t dev, uint32_t sm, uint32_t wp, *CuDim3* *blockIdx, bool *blockIdxValid)

Get the target block index and validity status for cluster exceptions.

Since CUDA 12.7.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param blockIdx
[out] - pointer to a *CuDim3* structure that will be populated with the target block index

Param blockIdxValid
[out] - pointer to a boolean variable that will be set to `true` if the target block index is valid, and `false` otherwise. Value will be set to false if the warp is not stopped on a cluster exception

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***getConstBankAddress**)(uint32_t dev, uint64_t gridId64, uint32_t bank, uint64_t *address, uint32_t *size)

Get constant bank GPU VA and size.

Since CUDA 12.4.

See also:

getBindlessConstAddress

Param dev
[in] - device index

Param gridId64
[in] - grid ID of the grid containing the constant bank

Param bank
[in] - constant bank number

Param address
[out] - GPU VA of the bank memory

Param size
[out] - bank size

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_MISSING_DATA

CUDBGResult (***getConstBankAddress123**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t bank, uint32_t offset, uint64_t *address)

Convert constant bank number and offset into GPU VA.

It's more convenient to get the constbank address and then calculate the VA for any const address using that.

Since CUDA 12.3.

See also:

getConstBankAddress

Note: DEPRECATED in CUDA 12.4: Use *getConstBankAddress* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param bank
[in] - constant bank number

Param offset
[in] - offset within the bank

Param address
[out] - GPU VA

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_MISSING_DATA

CUDBGResult (***getCudaExceptionString***)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, char *buf, uint32_t bufSz, uint32_t *msgSz)

Get the error string for CUDA Exceptions.

Since CUDA 13.0.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param buf
[out] - buffer for the error string

Param bufSz
[in] - buffer size

Param msgSz
[out] - error message size with null character, can be null

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_BUFFER_TOO_SMALL,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getDeviceInfo***)(uint32_t dev, *CUDBGDeviceInfoQueryType_t* type, void *buffer, uint32_t length, uint32_t *dataLength)

Read full device info for the device.

Information returned by this method is cheap to calculate, so it can be used after every suspend to quickly get the updated device state. For convenience, the caller can always request partial updates, the API will return a full response when returning a partial one is not possible.

If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 12.4.

See also:

getDeviceInfoSizes

Param dev
[in] - device index

Param type
[in] - query type (full or delta)

Param buffer
[out] - output buffer

Param length
[in] - output buffer length

Param dataLength
[out] - number of bytes written to the buffer

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*__getDeviceInfoSizes__)(uint32_t dev, *CUDBGDeviceInfoSizes* *sizes)

Return sizes for device info structs and defined attributes.

Since CUDA 12.4.

See also:

getDeviceInfo

Param dev
[in] - device index

Param sizes
[out] - device info sizes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

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

Get the device name string.

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 6.5.

See also:

getDeviceType

See also:

getSMType

Param dev
[in] - device index

Param buf
[out] - the destination buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_BUFFER_TOO_SMALL,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getDevicePCIBusInfo**)(uint32_t dev, uint32_t *pciBusId, uint32_t *pciDevId)

Get PCI bus and device ids associated with device index.

Since CUDA 5.5.

Param dev
[in] - device index

Param pciBusId
[out] - pointer where corresponding PCI BUS ID would be stored

Param pciDevId
[out] - pointer where corresponding PCI DEVICE ID would be stored

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getDeviceType**)(uint32_t dev, char *buf, uint32_t sz)

Get the string description of the device.

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getDeviceName

See also:

getSMType

Param dev
[in] - device index

Param buf
[out] - the destination buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_BUFFER_TOO_SMALL,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

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.

Since CUDA 4.0.

See also:

getElfImageByHandle

See also:

getLoadedFunctionInfo

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param relocated
[in] - set to true to specify the relocated ELF image, false otherwise

Param elfImage
[out] - pointer to the ELF image

Param size
[out] - size of the ELF image (64 bits)

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

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.

Behaves like getElfImage but will truncate the image size for cubins larger than 4GiB.

Since CUDA 3.0.

See also:

getElfImage

See also:

getElfImageByHandle

Note: DEPRECATED in CUDA 4.0: Use *getElfImage* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param relocated
[in] - set to true to specify the relocated ELF image, false otherwise

Param elfImage
[out] - pointer to the ELF image

Param size
[out] - size of the ELF image (32 bits)

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getElfImageByHandle**)(uint32_t dev, uint64_t handle, *CUDBGElfImageType* type, void *elfImage, uint64_t size)

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

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

Since CUDA 6.0.

See also:

getElfImage

See also:

getLoadedFunctionInfo

Param dev
[in] - device index

Param handle
[in] - elf image handle

Param type
[in] - type of the requested ELF image

Param elfImage
[out] - pointer to the ELF image

Param elfImage_size
[in] - size of the ELF image

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getErrorStringEx**)(char *buf, uint32_t bufSz, uint32_t *msgSz)

Fills a user-provided buffer with an error message encoded as a null-terminated ASCII string.

The error message is specific to the last failed API call and is invalidated after every API method call except this one. It's possible to query the size of the error message without reading it by passing 0 as `buf` and `bufSz` parameters. The `msgSz` parameter is optional unless 0 as passed in as `buf` and `bufSz`. CUDBG_ERROR_BUFFER_TOO_SMALL is returned when the passed in buffer is too small to contain the message.

Since CUDA 12.2.

Param buf
[out] - the destination buffer

Param bufSz
[in] - the size of the destination buffer in bytes

Param msgSz
[out] - the size of the written error message including the terminating null character.

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_BUFFER_TOO_SMALL,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getGridAttribute**)(uint32_t dev, uint32_t sm, uint32_t wp, *CUDBGAttribute* attr, uint64_t *value)

Get the value of a grid attribute.

See CUDBGAttribute for the list of available attributes.

Since CUDA 3.1.

See also:

getGridAttributes

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param attr
[in] - the attribute

Param value
[out] - the returned value of the attribute

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_ATTRIBUTE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

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.

See CUDBGAttribute for the list of available attributes.

Since CUDA 3.1.

See also:

getGridAttribute

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param pairs
[out] - array of attribute/value pairs

Param numPairs
[in] - the number of attribute/values pairs in the array

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_ATTRIBUTE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getGridDim**)(uint32_t dev, uint32_t sm, uint32_t wp, *CuDim3* *gridDim)

Get the dimensions in blocks of the given grid.

Since CUDA 4.0.

See also:

getBlockDim

See also:

getClusterDim

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param gridDim
[out] - the dimensions of the grid

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getGridDim32**)(uint32_t dev, uint32_t sm, uint32_t wp, *CuDim2* *gridDim)

Get the dimensions of the given grid.

Behaves like getGridDim but doesn't return the z dimension.

Since CUDA 3.0.

See also:

getGridDim

Note: DEPRECATED in CUDA 4.0: Use *getGridDim* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param gridDim
[out] - the returned number of blocks in the grid

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getGridInfo***)(uint32_t dev, uint64_t gridId64, *CUDBGGridInfo* *gridInfo)

Get information about the specified grid.

Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 12.7.

Param dev
[in] - device index

Param gridId
[in] - grid ID for which information is to be collected

Param gridInfo
[out] - pointer to a client allocated structure in which grid info will be returned

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getGridInfo120***)(uint32_t dev, uint64_t gridId64, *CUDBGGridInfo120* *gridInfo)

Get information about the specified grid.

Behaves like getGridInfo, but returns less information. Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 12.0.

See also:

getGridInfo

Note: DEPRECATED in CUDA 12.7: Use *getGridInfo* instead.

Param dev
[in] - device index

Param gridId
[in] - grid ID for which information is to be collected

Param gridInfo
[out] - pointer to a client allocated structure in which grid info will be returned

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getGridInfo55**)(uint32_t dev, uint64_t gridId64, *CUDBGGridInfo55* *gridInfo)

Get information about the specified grid.

Behaves like getGridInfo, but returns less information. Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 5.5.

See also:

getGridInfo

Note: DEPRECATED in CUDA 12.0: Use *getGridInfo* instead.

Param dev
[in] - device index

Param gridId
[in] - grid ID for which information is to be collected

Param gridInfo
[out] - pointer to a client allocated structure in which grid info will be returned

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getGridStatus**)(uint32_t dev, uint64_t gridId64, *CUDBGGridStatus* *status)

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

Since CUDA 5.5.

Param dev
[in] - device index

Param gridId64
[in] - 64-bit grid ID

Param status
[out] - enum indicating the grid status

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getGridStatus50**)(uint32_t dev, uint32_t gridId, *CUDBGGridStatus* *status)

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

Behaves like getGridStatus, but takes a 32-bit grid ID instead of a 64-bit one.

Since CUDA 5.0.

See also:

getGridStatus

Note: DEPRECATED in CUDA 5.5: Use *getGridStatus* instead.

Param dev
[in] - device index

Param gridId
[in] - grid ID

Param status
[out] - enum indicating the grid status

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getHardwareBarrierInfo**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, *CUDBGBarrierScope* *scope, char *buf, uint32_t bufSz, uint32_t *msgSz)

Get hardware barrier information.

Since CUDA 13.1.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param scope
[out] - barrier scope

Param buf
[out] - buffer for the barrier information

Param bufSz
[in] - buffer size

Param msgSz
[out] - error message size with null character, can be null

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_BUFFER_TOO_SMALL,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

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.

Since CUDA 4.1.

See also:

readGenericMemory

See also:

writeGenericMemory

Param dev
[in] - device index

Param device_addr
[in] - device memory address

Param host_addr
[out] - returned system memory address

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_SEGMENT,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getLoadedFunctionInfo**)(uint32_t dev, uint64_t handle, *CUDBGLoadedFunctionInfo* *info, uint32_t startIndex, uint32_t numEntries)

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

If the CUDBG_DEBUGGER_CAPABILITY_LAZY_FUNCTION_LOADING capability is enabled, CUDA loads functions lazily after the module has been reported. This method could be used to get the lazily loaded functions.

Since CUDA 12.3.

Param dev
[in] - device index

Param handle
[in] - ELF/cubin image handle

Param info
[out] - information about loaded functions

Param startIndex
[in] - start index of the entries to get

Param numEntries
[in] - number of function load entries to read

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getLoadedFunctionInfo118**)(uint32_t dev, uint64_t handle, *CUDBGLoadedFunctionInfo* *info, uint32_t numEntries)

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

Behaves like getLoadedFunctionInfo but doesn't allow querying a subset of all lazily loaded functions.

Since CUDA 11.8.

See also:

getLoadedFunctionInfo

Note: DEPRECATED in CUDA 12.3: Use *getLoadedFunctionInfo* instead.

Param dev
[in] - device index

Param handle
[in] - ELF/cubin image handle

Param info
[out] - information about loaded functions

Param numEntries
[in] - number of function load entries to read

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getManagedMemoryRegionInfo***)(uint64_t startAddress, *CUDBGMemoryInfo* *memoryInfo, uint32_t memoryInfo_size, uint32_t *numEntries)

Get a sorted list of managed memory regions.

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

Since CUDA 6.0.

Param startAddress
[in] - the address that the first region in the list must contain

Param memoryInfo
[out] - client-allocated array of memory region records of type *CUDBGMemoryInfo*

Param memoryInfo_size
[in] - number of records of type *CUDBGMemoryInfo* that memoryInfo can hold

Param numEntries
[out] - pointer to a client-allocated variable holding the number of valid entries returned in memoryInfo

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getNextAsyncEvent50**)(*CUDBGEvent50* *event)

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

Behaves like getNextEvent but only for ASYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.0.

See also:

getNextEvent

Note: DEPRECATED in CUDA 5.5: Use *getNextEvent* instead.

Param event
[out] - pointer to an event container where to copy the event parameters

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_INVALID_CONTEXT

CUDBGResult (***getNextAsyncEvent55**)(*CUDBGEvent55* *event)

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

Behaves like getNextEvent but only for ASYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.5.

See also:

getNextEvent

Note: DEPRECATED in CUDA 6.0: Use *getNextEvent* instead.

Param event
[out] - pointer to an event container where to copy the event parameters

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_INVALID_CONTEXT

CUDBGResult (***getNextEvent**)(*CUDBGEventQueueType* type, *CUDBGEvent* *event)

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

CUDBG_ERROR_NO_EVENT_AVAILABLE is returned if the queue is empty. ASYNC and SYNC queues are separate and each one is ordered separately, but it's impossible to find out the relative order of ASYNC and SYNC events.

Since CUDA 6.0.

See also:

acknowledgeSyncEvents

See also:

setNotifyNewEventCallback

Param type
[in] - application event queue type

Param event
[out] - pointer to an event container where to copy the event parameters

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE

CUDBGResult (***getNextEvent30**)(*CUDBGEvent30* *event)

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

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 3.0.

See also:

getNextEvent

Note: DEPRECATED in CUDA 3.1: Use *getNextEvent* instead.

Param event
[out] - pointer to an event container where to copy the event parameters

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_INVALID_CONTEXT

CUDBGResult (***getNextEvent32**)(*CUDBGEvent32* *event)

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

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 3.1.

See also:

getNextEvent

Note: DEPRECATED in CUDA 4.0: Use *getNextEvent* instead.

Param event
[out] - pointer to an event container where to copy the event parameters

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_INVALID_CONTEXT

CUDBGResult (***getNextEvent42**)(*CUDBGEvent42* *event)

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

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 4.0.

See also:

getNextEvent

Note: DEPRECATED in CUDA 5.0: Use *getNextEvent* instead.

Param event
[out] - pointer to an event container where to copy the event parameters

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_INVALID_CONTEXT

CUDBGResult (***getNextSyncEvent50**)(*CUDBGEvent50* *event)

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

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.0.

See also:

getNextEvent

Note: DEPRECATED in CUDA 5.5: Use *getNextEvent* instead.

Param event
[out] - pointer to an event container where to copy the event parameters

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_INVALID_CONTEXT

CUDBGResult (***getNextSyncEvent55**)(*CUDBGEvent55* *event)

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

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.5.

See also:

getNextEvent

Note: DEPRECATED in CUDA 6.0: Use *getNextEvent* instead.

Param event
[out] - pointer to an event container where to copy the event parameters

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_EVENT_AVAILABLE,

Return
CUDBG_ERROR_INVALID_CONTEXT

CUDBGResult (***getNumDevices**)(uint32_t *numDev)

Get the number of installed CUDA devices.

This value is constant within a single session.

Since CUDA 3.0.

See also:

getNumLanes

See also:

getNumPredicates

See also:

getNumRegisters

See also:

getNumSMs

See also:

getNumUniformPredicates

See also:

getNumUniformRegisters

See also:

getNumWarps

Param numDev
[out] - the returned number of devices

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getNumLanes**)(uint32_t dev, uint32_t *numLanes)

Get the number of lanes per warp on the device.

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

See also:

getNumPredicates

See also:

getNumRegisters

See also:

getNumSMs

See also:

getNumUniformPredicates

See also:

getNumUniformRegisters

See also:

getNumWarps

Param dev
[in] - device index

Param numLanes
[out] - the returned number of lanes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getNumPredicates**)(uint32_t dev, uint32_t *numPredicates)

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

This value is constant within a single session for a given device.

Since CUDA 6.5.

See also:

getNumDevices

See also:

getNumLanes

See also:

getNumRegisters

See also:

getNumSMs

See also:

getNumUniformPredicates

See also:

getNumUniformRegisters

See also:

getNumWarps

Param dev
[in] - device index

Param numPredicates
[out] - the returned number of predicate registers

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getNumRegisters**)(uint32_t dev, uint32_t *numRegs)

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

This value is constant within a single session for a given device. Note that the actual number of registers can change per warp, use *readWarpResources()* to query that number dynamically.

Since CUDA 3.0.

See also:

getNumDevices

See also:

getNumLanes

See also:

getNumPredicates

See also:

getNumSMs

See also:

getNumUniformPredicates

See also:

getNumUniformRegisters

See also:

getNumWarps

See also:

readWarpResources

Param dev
[in] - device index

Param numRegs
[out] - the returned number of registers

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getNumSMs**)(uint32_t dev, uint32_t *numSMs)

Get the total number of SMs on the device.

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

See also:

getNumLanes

See also:

getNumPredicates

See also:

getNumRegisters

See also:

getNumUniformPredicates

See also:

getNumUniformRegisters

See also:

getNumWarps

Param dev
[in] - device index

Param numSMs
[out] - the returned number of SMs

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getNumUniformPredicates**)(uint32_t dev, uint32_t *numPredicates)

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

This value is constant within a single session for a given device.

Since CUDA 10.0.

See also:

getNumDevices

See also:

getNumLanes

See also:

getNumPredicates

See also:

getNumRegisters

See also:

getNumSMs

See also:

getNumUniformRegisters

See also:

getNumWarps

Param dev
[in] - device index

Param numPredicates
[out] - the returned number of uniform predicate registers

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getNumUniformRegisters**)(uint32_t dev, uint32_t *numRegs)

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

This value is constant within a single session for a given device.

Since CUDA 10.0.

See also:

getNumDevices

See also:

getNumLanes

See also:

getNumPredicates

See also:

getNumRegisters

See also:

getNumSMs

See also:

getNumUniformPredicates

See also:

getNumWarps

Param dev
[in] - device index

Param numRegs
[out] - the returned number of uniform registers

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getNumWarps**)(uint32_t dev, uint32_t *numWarps)

Get the number of warps per SM on the device.

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

See also:

getNumLanes

See also:

getNumPredicates

See also:

getNumRegisters

See also:

getNumSMs

See also:

getNumUniformPredicates

See also:

getNumUniformRegisters

Param dev
[in] - device index

Param numWarps
[out] - the returned number of warps

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getPhysicalRegister30**)(uint64_t pc, char *reg, uint32_t *buf, uint32_t sz, uint32_t *numPhysRegs, *CUDBGRegClass* *regClass)

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

Since CUDA 3.0.

Note: DEPRECATED in CUDA 3.1: Do not use.

Param pc
[in] - Program counter

Param reg
[in] - virtual register index

Param buf
[out] - physical register name(s)

Param sz
[in] - the physical register name buffer size

Param numPhysRegs
[out] - number of physical register names returned

Param regClass
[out] - the class of the physical registers

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_BUFFER_TOO_SMALL,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

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 at a given PC, if it's live at that PC.

Instead, the PTX to SASS mappings can be read from the cubin directly.

Since CUDA 3.1.

Note: DEPRECATED in CUDA 4.1: Do not use.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param pc
[in] - Program counter

Param reg
[in] - virtual register index

Param buf
[out] - physical register name(s)

Param sz
[in] - the physical register name buffer size

Param numPhysRegs
[out] - number of physical register names returned

Param regClass
[out] - the class of the physical registers

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_BUFFER_TOO_SMALL,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getSmType**)(uint32_t dev, char *buf, uint32_t sz)

Get the SM type of the device.

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getDeviceName

See also:

getDeviceType

Param dev
[in] - device index

Param buf
[out] - the destination buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_BUFFER_TOO_SMALL,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getSupportedDebuggerCapabilities**)(*CUDBGCapabilityFlags* *capabilities)

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

This API method can be called without initializing the API.

Since CUDA 12.5.

Param capabilities
[out] - returned debug engine capabilities

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getTID**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *tid)

Get the ID of the Linux thread hosting the CUDA context active at the given coordinates.

This returns a Linux thread ID.

Since CUDA 3.0.

See also:

getGridAttributes

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param tid
[out] - the returned thread id

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***getWarpHitBreakpoint**)(uint32_t dev, uint32_t sm, uint32_t wp, *CUDBGBreakpointHandle* *handle)

Get the handle of the breakpoint that the given warp hit.

An error is returned if the warp did not hit a breakpoint. Use *readBrokenWarps()* to check if the warp is broken before calling this method. Some breakpoint handles are special, see the documentation of CUDBGBreakpointHandle for more details.

Since CUDA 13.2.

See also:

disableBreakpoint

See also:

enableBreakpoint

See also:

insertBreakpoint

See also:

isBreakpointEnabled

See also:

readBrokenWarps

See also:

removeBreakpoint

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param handle
[out] - the returned breakpoint handle

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***initialize**)(void)

Initialize the API.

setNotifyNewEventCallback() and *getSupportedDebuggerCapabilities()* can be called before *initialize()*. If no CUDA devices are detected on the system, CUDBG_ERROR_NO_DEVICE_AVAILABLE is returned.

Since CUDA 3.0.

See also:

finalize

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_NO_DEVICE_AVAILABLE

CUDBGResult (***initializeAttachStub**)(void)

Initialize the attach stub.

This is no longer necessary starting with driver version r590.

Since CUDA 5.0.

Return
CUDBG_SUCCESS

CUDBGResult (***insertBreakpoint**)(uint32_t dev, uint64_t addr, *CUDBGBreakpointHandle* *handle)

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

Before setting a breakpoint, *getAdjustedCodeAddress()* should be called to get the adjusted breakpoint address. The returned handle can be used to enable/disable/remove the breakpoint.

Since CUDA 13.2.

See also:

disableBreakpoint

See also:

enableBreakpoint

See also:

getWarpHitBreakpoint

See also:

isBreakpointEnabled

See also:

removeBreakpoint

Param dev
[in] - the device index

Param addr
[in] - instruction address

Param handle
[out] - the returned breakpoint handle

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

CUDBGResult (***isBreakpointEnabled**)(*CUDBGBreakpointHandle* handle, uint32_t *enabled)

Check if a breakpoint specified by its handle is enabled.

The breakpoint enablement state is never implicitly modified by the API implementation; the API user must always enable or disable them manually. Prior to CUDA 13.4, an error is returned if the CUDBG_BREAKPOINT_HANDLE_BREAK_ON_LAUNCH handle is provided.

Since CUDA 13.2.

See also:

disableBreakpoint

See also:

enableBreakpoint

See also:

getWarpHitBreakpoint

See also:

insertBreakpoint

See also:

removeBreakpoint

Param handle
[in] - the breakpoint handle.

Param enabled
[out] - whether the breakpoint is enabled

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***isDeviceCodeAddress**)(uintptr_t addr, bool *isDeviceAddress)

Determine whether a virtual address resides within device code.

Since CUDA 3.0.

Param addr
[in] - virtual address

Param isDeviceAddress
[out] - true if address resides within device code

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***isDeviceCodeAddress55**)(uintptr_t addr, bool *isDeviceAddress)

Determine whether a virtual address resides within device code.

Behaves exactly like isDeviceCodeAddress.

Since CUDA 3.0.

See also:

isDeviceCodeAddress

Note: DEPRECATED in CUDA 6.0: Use *isDeviceCodeAddress* instead.

Param addr
[in] - virtual address

Param isDeviceAddress
[out] - true if address resides within device code

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***lookupDeviceCodeSymbol**)(char *symName, bool *symFound, uintptr_t *symAddr)

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

Since CUDA 3.0.

Param symName
[in] - symbol name

Param symFound
[out] - set to true if the symbol is found

Param symAddr
[out] - the symbol virtual address if found

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***memcheckReadErrorAddress**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *address, *ptxStorageKind* *storage)

Get the address that memcheck detected an error on.

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 5.0.

Note: DEPRECATED in CUDA 12.0: Do not use.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param address
[out] - returned address detected by memcheck

Param storage
[out] - returned address class of address

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***readActiveLanes**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *activeLanesMask)

Read the lane bitmask of active threads on a valid warp.

Since CUDA 3.0.

See also:

readBlockIdx

See also:

readBrokenWarps

See also:

readGridId

See also:

readThreadIdx

See also:

readValidLanes

See also:

readValidWarps

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param activeLanesMask
[out] - the returned bitmask of active threads

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readAllVirtualReturnAddresses**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *addrs, uint32_t numAddrs, uint32_t *callDepth, uint32_t *syscallCallDepth)

Read all the virtual return addresses for a thread (the full backtrace).

Note that syscallCallDepth is always set to 0.

Since CUDA 12.5.

See also:

readCallDepth

See also:

readReturnAddress

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param addrs
[out] - the returned addresses array

Param numAddrs
[in] - number of elements in addrs array

Param callDepth
[out] - the returned call depth

Param syscallCallDepth
[out] - the returned syscall call depth

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readBlockIdx**)(uint32_t dev, uint32_t sm, uint32_t wp, *CuDim3* *blockIdx)

Read the CUDA block index running on a valid warp.

Since CUDA 4.0.

See also:

readActiveLanes

See also:

readBrokenWarps

See also:

readGridId

See also:

readThreadIdx

See also:

readValidLanes

See also:

readValidWarps

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param blockIdx
[out] - the returned CUDA block index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`readBlockIdx32`)(uint32_t dev, uint32_t sm, uint32_t wp, *CuDim2* *blockIdx)

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

Behaves like readBlockIdx but doesn't return the z dimension.

Since CUDA 3.0.

See also:

readBlockIdx

Note: DEPRECATED in CUDA 4.0: Use *readBlockIdx* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param blockIdx
[out] - the returned CUDA block index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readBrokenWarps**)(uint32_t dev, uint32_t sm, uint64_t *brokenWarpsMask)

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

Since CUDA 3.0.

See also:

readActiveLanes

See also:

readBlockIdx

See also:

readGridId

See also:

readThreadIdx

See also:

readValidLanes

See also:

readValidWarps

Param dev
[in] - device index

Param sm
[in] - SM index

Param brokenWarpsMask
[out] - the returned bitmask of broken warps

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readCallDepth**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *depth)

Read the call depth (number of calls) for a given thread.

Since CUDA 4.0.

See also:

readReturnAddress

See also:

readVirtualReturnAddress

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param depth
[out] - the returned call depth

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readCallDepth32**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *depth)

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

Behaves like *readCallDepth()* for the active thread group.

Since CUDA 3.1.

See also:

readCallDepth

Note: DEPRECATED in CUDA 4.0: Use *readCallDepth* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param depth
[out] - the returned call depth

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_LANE,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readCCRegister**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *val)

Read the hardware CC register.

The CC register is no longer available in the supported hardware.

Since CUDA 6.5.

Note: DEPRECATED in CUDA 13.1: Do not use.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param val
[out] - the returned value of the CC register

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readClusterIdx**)(uint32_t dev, uint32_t sm, uint32_t wp, *CuDim3* *clusterIdx)

Read the CUDA cluster index running on a valid warp.

Since CUDA 12.0.

See also:
readActiveLanes

See also:
readBlockIdx

See also:
readBrokenWarps

See also:
readGridId

See also:
readThreadIdx

See also:
readValidLanes

See also:
readValidWarps

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param clusterIdx
[out] - the returned CUDA cluster index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`readCodeMemory`)(uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the code memory segment.

It is generally not necessary to call this function - instead, the same memory could be read from the ELF module images received from the API.

Since CUDA 3.0.

See also:

readGenericMemory

See also:

readLocalMemory

See also:

readPC

See also:

readParamMemory

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

Param dev
[in] - device index

Param addr
[in] - memory address

Param buf
[out] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*__readConstMemory129__)(uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the constant memory segment.

Behaves exactly like readGlobalMemory.

Since CUDA 3.0.

See also:

readGlobalMemory

Note: DEPRECATED in CUDA 13.0: Use *readGlobalMemory* instead.

Param dev
[in] - device index

Param addr
[in] - memory address

Param buf
[out] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***readCPUCallStack**)(uint32_t dev, uint64_t gridId64, uint64_t *addrs, uint32_t numAddrs, uint32_t *totalNumAddrs)

Read the CPU call stack captured at the time of kernel launch.

This method only works if the CUDBG_DEBUGGER_CAPABILITY_COLLECT_CPU_CALL_STACK_FOR_KERNEL_ capability is enabled.

Since CUDA 12.9.

Param dev
[in] - device index

Param gridId64
[in] - 64-bit grid ID

Param addrs
[out] - the returned addresses array, can be NULL

Param numAddrs
[in] - capacity of addrs (possibly 0)

Param totalNumAddrs
[out] - the actual size of the stack (number of frames) is written here; the value written can be greater than numAddrs

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`readDeviceExceptionState`)(uint32_t dev, uint64_t *mask, uint32_t numWords)

Get the exception state of the SMs on the device.

If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 9.0.

See also:

getNumSMs

Param dev
[in] - device index

Param mask
[out] - Arbitrarily sized bit field containing a 1 at (1 << i) if SM i hit an exception

Param numWords
[in] - Number of uint64_t elements in `mask` (must be large enough to hold a bit for each sm on the device)

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`readDeviceExceptionState80`)(uint32_t dev, uint64_t *exceptionSMMask)

Get the exception state of the SMs on the device.

Behaves like readDeviceExceptionState but only supports up to 64 SMs.

Since CUDA 5.5.

See also:

readDeviceExceptionState

Note: DEPRECATED in CUDA 9.0: Use *readDeviceExceptionState* instead.

Param dev
[in] - device index

Param exceptionSMMask
[out] - Bit field containing a 1 at (1 << i) if SM i hit an exception

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readErrorPC**)(uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *errorPC, bool *errorPCValid)

Get the hardware reported error PC if it exists.

The error PC, if available, shows the PC where an error happened (the thread can progress past that so its PC could be beyond that).

Since CUDA 6.0.

Param dev
[in] - device index

Param sm
[in] - SM index

Param warp
[in] - warp index

Param errorPC
[out] - PC ofthe exception

Param errorPCValid
[out] - boolean to indicate that the returned error PC is valid

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readGenericMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at an address in any memory segment.

The address will be used to determine whether the read is to local, shared or global memory. The target address range should entirely reside within a single memory segment. Coordinate arguments are only used when relevant. They should be provided for the following segments:

- Shared memory: SM and Warp
- Local memory: SM, Warp and Lane

Since CUDA 6.0.

See also:

readCodeMemory

See also:

readLocalMemory

See also:

readPC

See also:

readParamMemory

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param addr
[in] - memory address

Param buf
[out] - buffer

Param buf_size
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (*`readGlobalMemory`)(uint64_t addr, void *buf, uint32_t sz)

Read content at an address in the global address space.

If the address is valid on more than one device and one of those devices does not support UVA, an error is returned.

Since CUDA 6.0.

See also:

readCodeMemory

See also:

readLocalMemory

See also:

readPC

See also:

readParamMemory

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

Param addr
[in] - memory address

Param buf
[out] - buffer

Param buf_size
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***readGlobalMemory31**)(uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the global memory segment.

Behaves like *readGenericMemory()* with sm, wp, ln == 0. This makes this method not at all useful.

Since CUDA 3.0.

See also:

readGlobalMemory

Note: DEPRECATED in CUDA 3.2: Use *readGlobalMemory* instead.

Param dev
[in] - device index

Param addr
[in] - memory address

Param buf
[out] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***readGlobalMemory55**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the global memory segment.

Behaves exactly like *readGenericMemory()*.

Since CUDA 3.2.

See also:

readGlobalMemory

Note: DEPRECATED in CUDA 6.0: Use *readGlobalMemory* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param addr
[in] - memory address

Param buf
[out] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (*__readGridId__)(uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *gridId64)

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

The grid ID is guaranteed to be unique within a device, but not globally.

Since CUDA 5.5.

See also:

readActiveLanes

See also:

readBlockIdx

See also:

readBrokenWarps

See also:

readThreadIdx

See also:

readValidLanes

See also:

readValidWarps

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param gridId
[out] - the returned 64-bit CUDA grid index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readGridId50**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *gridId)

Read the CUDA grid index running on a valid warp.

Behaves like readGridId but truncates the grid ID to 32bit. This is incompatible with some grid IDs like those used by the OptiX applications.

Since CUDA 3.0.

See also:

readGridId

Note: DEPRECATED in CUDA 5.5: Use *readGridId* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param gridId
[out] - the returned CUDA grid index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readLaneException**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CUDBGException_t *exception)

Read the exception type for a given thread.

Since CUDA 3.1.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param exception
[out] - the returned exception type

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readLaneStatus**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, bool *error)

Read the status of the given thread.

For specific error values, use readLaneException.

Since CUDA 3.0.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param error
[out] - true if there is an error

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readLocalMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the local memory segment.

Since CUDA 3.0.

See also:

readCodeMemory

See also:

readGenericMemory

See also:

readPC

See also:

readParamMemory

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param addr
[in] - memory address

Param buf
[out] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`readParamMemory`)(uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the param memory segment.

Since CUDA 3.0.

See also:

readCodeMemory

See also:

readGenericMemory

See also:

readLocalMemory

See also:

readPC

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param addr
[in] - memory address

Param buf
[out] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readPC**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *pc)

Read the PC offset on the given active thread.

The returned PC offset is from the start of the current function. If a function can't be found, the full virtual address is returned.

Since CUDA 3.0.

See also:

readCodeMemory

See also:

readGenericMemory

See also:

readLocalMemory

See also:

readParamMemory

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

See also:

readVirtualPC

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param pc
[out] - the returned PC

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*__readPinnedMemory__)(uint64_t addr, void *buf, uint32_t sz)

Read content at pinned address in system memory.

Depending on the platform, this method may fail and a platform-specific CPU RAM way of reading memory from the debuggee must be used (e.g. ptrace).

Since CUDA 3.2.

See also:

readCodeMemory

See also:

readGenericMemory

See also:

readLocalMemory

See also:

readPC

See also:

readParamMemory

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

Param addr
[in] - system memory address

Param buf
[out] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_MEMORY_MAPPING_FAILED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readPredicates**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t predicates_size, uint32_t *predicates)

Read content of hardware predicate registers.

Since CUDA 6.5.

See also:

readCodeMemory

See also:

readGenericMemory

See also:

readGlobalMemory

See also:

readLocalMemory

See also:

readPC

See also:

readParamMemory

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param predicates_size
[in] - number of predicate registers to read

Param predicates
[out] - buffer

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readRegister**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t regno, uint32_t *val)

Read content of a hardware register.

Note that warps can dynamically change the number of used registers at runtime, *readWarpResources()* could be used to query that.

Since CUDA 3.0.

See also:

readCodeMemory

See also:

readGenericMemory

See also:

readLocalMemory

See also:

readPC

See also:

readParamMemory

See also:

readSharedMemory

See also:

readTextureMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param regno
[in] - register index

Param val
[out] - the returned value of the register

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readRegisterRange**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t index, uint32_t numRegisters, uint32_t *registers, uint32_t *numRegistersRead)

Read content of a hardware range of hardware registers.

Since CUDA 13.2.

See also:

readCodeMemory

See also:

readGenericMemory

See also:

readLocalMemory

See also:

readPC

See also:

readParamMemory

See also:

readRegister

See also:

readSharedMemory

See also:

readTextureMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param index
[in] - index of the first register to read

Param numRegisters
[in] - number of registers to read

Param registers
[out] - buffer

Param numRegistersRead
[out] - number of registers actually read, ignored if null

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readRegisterRange60**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t index, uint32_t registers_size, uint32_t *registers)

Read content of a range of hardware registers.

Since CUDA 6.0.

See also:

readRegisterRange

Note: DEPRECATED in CUDA 13.2: Use *readRegisterRange* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param index
[in] - index of the first register to read

Param registers_size
[in] - number of registers to read

Param registers
[out] - buffer

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readReturnAddress**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t level, uint64_t *ra)

Read the return address (offset) for a call level.

The returned return address is an offset from the start of the current function. If a function can't be found, the full virtual address is returned.

Since CUDA 4.0.

See also:

readCallDepth

See also:

readVirtualReturnAddress

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param level
[in] - the specified call level

Param ra
[out] - the returned return address for level

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CALL_LEVEL,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readReturnAddress32**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t level, uint64_t *ra)

Read the return address (offset) for a call level.

Behaves like *readReturnAddress()* for the active thread group.

Since CUDA 3.1.

See also:

readReturnAddress

Note: DEPRECATED in CUDA 4.0: Use *readReturnAddress* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param level
[in] - the specified call level

Param ra
[out] - the returned return address for level

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_LANE,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CALL_LEVEL,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readRpcRegisters**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *rpcLo, uint32_t *rpcHi)

Read the RPC.LO and/or RPC.HI hardware registers for a specific thread.

These registers are normally used by hardware to store the program counter of diverged threads. In some cases the compiler may also spill data into them for active threads. These registers can be read for both active and diverged threads.

Passing NULL for `rpcLo` or `rpcHi` causes that register to be skipped. Passing NULL for both is an error.

Since CUDA 13.4.

See also:

writeRpcRegisters

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param rpcLo
[out] - receives RPC.LO (lower 32 bits), or NULL to skip

Param rpcHi
[out] - receives RPC.HI (upper 32 bits), or NULL to skip

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INVALID_DEVICE,

Return
CUDBG_ERROR_INVALID_SM,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_LANE,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readSharedMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the shared memory segment.

Since CUDA 3.0.

See also:

readCodeMemory

See also:

readGenericMemory

See also:

readLocalMemory

See also:

readPC

See also:

readParamMemory

See also:

readRegister

See also:

readTextureMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param addr
[in] - memory address

Param buf
[out] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readSmException***)(uint32_t dev, uint32_t sm, CUDBGException_t *exception, uint64_t *errorPC, bool *errorPCValid)

Get the SM exception status if it exists.

If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 12.5.

Param dev
[in] - the device index

Param sm
[in] - the SM index

Param exception
[out] - returned exception

Param errorPC
[out] - returned PC of the exception

Param errorPCValid
[out] - boolean to indicate that the returned error PC is valid

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readSyscallCallDepth***)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *depth)

Read the call depth of syscalls for a given thread.

Will always return 0.

Since CUDA 4.1.

Note: DEPRECATED in CUDA 12.9: Do not use.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param depth
[out] - the returned call depth

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readTcgen05TmemInfo**)(uint32_t dev, uint32_t sm, uint32_t wp, *CUDBGTcgen05TmemInfo* *info)

Read tcgen05 Tensor Memory metadata for the CTA containing a warp.

Since CUDA 13.5.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param info
[out] - receives tcgen05 Tensor Memory metadata

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_NOT_SUPPORTED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readTcgen05TmemMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t addr, void *buf, uint32_t sz)

Read tcgen05 Tensor Memory data for the CTA containing a warp.

Since CUDA 13.5.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param addr
[in] - Tensor Memory address: bits 31:16 are lane, bits 15:0 are column

Param buf
[out] - buffer that receives Tensor Memory data

Param sz
[in] - number of bytes to read

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_NOT_SUPPORTED,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readTextureMemory**)(uint32_t dev, uint32_t vsm, uint32_t wp, uint32_t id, uint32_t dim, uint32_t *coords, void *buf, uint32_t sz)

This method is no longer supported since CUDA 12.0.

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 4.0.

Note: DEPRECATED in CUDA 12.0: Do not use.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param id
[in] - texture id (the value of DW_AT_location attribute in the relocated ELF image)

Param dim
[in] - texture dimension (1 to 4)

Param coords
[in] - array of coordinates of size dim

Param buf
[out] - result buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***readTextureMemoryBindless**)(uint32_t dev, uint32_t vsm, uint32_t wp, uint32_t texSymtabIndex, uint32_t dim, uint32_t *coords, void *buf, uint32_t sz)

This method is no longer supported since CUDA 12.0.

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 4.2.

Note: DEPRECATED in CUDA 12.0: Do not use.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param texSymtabIndex
[in] - global symbol table index of the texture symbol

Param dim
[in] - texture dimension (1 to 4)

Param coords
[in] - array of coordinates of size dim

Param buf
[out] - result buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***readThreadIdx**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, *CuDim3* *threadIdx)

Read the CUDA thread index running on valid thread.

Since CUDA 3.0.

See also:

readActiveLanes

See also:

readBlockIdx

See also:

readBrokenWarps

See also:

readGridId

See also:

readValidLanes

See also:

readValidWarps

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param threadIdx
[out] - the returned CUDA thread index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readUniformPredicates**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t predicates_size, uint32_t *predicates)

Read contents of uniform predicate registers.

Since CUDA 10.0.

See also:

readPredicates

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param predicates_size
[in] - number of predicate registers to read

Param predicates
[out] - buffer

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readUniformRegisterRange**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t regno, uint32_t registers_size, uint32_t *registers)

Read a range of uniform registers.

Since CUDA 10.0.

See also:

readRegister

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param regno
[in] - starting index into uniform register file

Param registers_size
[in] - number of bytes to read

Param registers
[out] - pointer to buffer

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`readValidLanes`)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *validLanesMask)

Read the lane bitmask of valid threads on a given warp.

Since CUDA 3.0.

See also:

readActiveLanes

See also:

readBlockIdx

See also:

readBrokenWarps

See also:

readGridId

See also:

readThreadIdx

See also:

readValidWarps

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param validLanesMask
[out] - the returned bitmask of valid threads

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readValidWarps**)(uint32_t dev, uint32_t sm, uint64_t *validWarpsMask)

Read the bitmask of valid warps on a given SM.

Since CUDA 3.0.

See also:

readActiveLanes

See also:

readBlockIdx

See also:

readBrokenWarps

See also:

readGridId

See also:

readThreadIdx

See also:

readValidLanes

Param dev
[in] - device index

Param sm
[in] - SM index

Param validWarpsMask
[out] - the returned bitmask of valid warps

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`readVirtualPC`)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *pc)

Read the PC (virtual address) on the given active thread.

Since CUDA 3.0.

See also:

readPC

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param pc
[out] - the returned PC

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readVirtualReturnAddress**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t level, uint64_t *ra)

Read the virtual return address for a call level.

Since CUDA 4.0.

See also:

readCallDepth

See also:

readReturnAddress

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param level
[in] - the specified call level

Param ra
[out] - the returned virtual return address for level

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CALL_LEVEL,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*__readVirtualReturnAddress32__)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t level, uint64_t *ra)

Read the virtual return address for a call level.

Behaves like readVirtualReturnAddress for the active thread group.

Since CUDA 3.1.

See also:

readVirtualReturnAddress

Note: DEPRECATED in CUDA 4.0: Use *readVirtualReturnAddress* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param level
[in] - the specified call level

Param ra
[out] - the returned virtual return address for level

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_LANE,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CALL_LEVEL,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readWarpResources**)(uint32_t dev, uint32_t sm, uint32_t wp, *CUDBGWarpResources* *resources)

Get the resources assigned to a given warp.

Note that these resources can change between suspends, which makes this method useful for avoiding warp data access errors.

Since CUDA 12.8.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param resources
[out] - pointer to structure that contains warp resources

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readWarpState**)(uint32_t dev, uint32_t sm, uint32_t wp, *CUDBGWarpState* *state)

Read the state of a given warp.

Since CUDA 12.9.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param state
[out] - pointer to structure that contains warp state

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readWarpState120**)(uint32_t dev, uint32_t sm, uint32_t wp, *CUDBGWarpState120* *state)

Read the state of a given warp.

Behaves like readWarpState but returns fewer fields.

Since CUDA 12.0.

See also:

readWarpState

Note: DEPRECATED in CUDA 12.7: Use *readWarpState* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param state
[out] - pointer to structure that contains warp state

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readWarpState127**)(uint32_t dev, uint32_t sm, uint32_t wp, *CUDBGWarpState127* *state)

Read the state of a given warp.

Behaves like readWarpState but returns fewer fields.

Since CUDA 12.7.

See also:

readWarpState

Note: DEPRECATED in CUDA 12.9: Use *readWarpState* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param state
[out] - pointer to structure that contains warp state

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***readWarpState60**)(uint32_t dev, uint32_t sm, uint32_t wp, *CUDBGWarpState60* *state)

Read the state of a given warp.

Behaves like readWarpState but returns fewer fields.

Since CUDA 6.0.

See also:

readWarpState

Note: DEPRECATED in CUDA 12.0: Use *readWarpState* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param state
[out] - pointer to structure that contains warp state

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_GRID,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***removeBreakpoint**)(*CUDBGBreakpointHandle* handle)

Remove a breakpoint specified by its handle.

Since CUDA 13.2.

See also:

disableBreakpoint

See also:

enableBreakpoint

See also:

getWarpHitBreakpoint

See also:

insertBreakpoint

See also:

isBreakpointEnabled

Param handle
[in] - the breakpoint handle. If it's *CUDBG_BREAKPOINT_HANDLE_ALL_USER_BREAKPOINTS*, remove all breakpoints inserted by the client (except the break-on-launch breakpoint). It's fine to use ALL_USER_BREAKPOINTS when there are no breakpoints added.

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***requestCleanupOnDetach**)(uint32_t appResumeFlag)

Request for cleanup of driver state when detaching.

Needs to be conditionally called by the client depending on the state of the debugged application. See the “Attaching and Detaching” section for more information.

Since CUDA 6.0.

Param appResumeFlag
[in] - value of CUDBG_RESUME_FOR_ATTACH_DETACH as read from the application’s process space.

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***requestCleanupOnDetach55**)(void)

Request for cleanup of driver state when detaching.

Needs to be conditionally called by the client depending on the state of the debugged application. See the “Attaching and Detaching” section for more information.

Since CUDA 5.0.

See also:

requestCleanupOnDetach

Note: DEPRECATED in CUDA 6.0: Use *requestCleanupOnDetach* instead.

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`resumeAllDevices`)()

Resume all running CUDA devices.

Since CUDA 13.2.

See also:

suspendAllDevices

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_RUNNING_DEVICE,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`resumeDevice`)(uint32_t dev)

Resume a suspended CUDA device.

Using this method is discouraged, use *resumeAllDevices()* instead to avoid race conditions. This method has no effect if the device is already running.

Since CUDA 3.0.

See also:

resumeAllDevices

Param dev
[in] - device index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_RUNNING_DEVICE,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***resumeWarpsUntilPC**)(uint32_t dev, uint32_t sm, uint64_t warpMask, uint64_t pc, uint32_t flags)

Insert a temporary breakpoint at the specified virtual PC and resume all warps in the specified bitmask on a given SM.

Compared to *resumeDevice()*, this method provides finer-grain control by resuming a selected set of warps on the same SM. The main intended usage is to accelerate the single-stepping process when the target PC is known in advance. Instead of single-stepping each warp individually until the target PC is hit, the client can use this method. If an unsteppable barrier is hit by the resumed warps, this method returns early (before reaching the target PC). When this method is used, errors within CUDA kernels will no longer be reported precisely. In the situation where resuming warps is not possible, this method will return CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE. The client should then fall back to using *singleStepWarp()* or *resumeDevice()*.

This method is non-blocking if the CUDBG_SINGLE_STEP_FLAGS_NON_BLOCKING flag is passed in, otherwise it blocks until the operation is complete. The CUDBG_EVENT_SINGLE_STEP_COMPLETE event is only sent for the non-blocking mode. If a non-blocking operation is interrupted by a breakpoint or an exception, and the step is not complete, then CUDBG_EVENT_ALL_DEVICES_SUSPENDED is sent instead.

Since CUDA 13.2.

See also:

resumeAllDevices

See also:

singleStepWarp

Param dev
[in] - device index

Param sm
[in] - the SM index

Param warpMask
[in] - the bitmask of warps to resume (1 = resume, 0 = do not resume)

Param virtPC
[in] - the virtual PC where the temporary breakpoint will be inserted

Param flags
[in] - flags of type CUDBGSingleStepFlags to change the stepping behavior

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_RUNNING_DEVICE,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE,

Return
CUDBG_ERROR_INVALID_WARP_MASK,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*__resumeWarpsUntilPC60__)(uint32_t dev, uint32_t sm, uint64_t warpMask, uint64_t virtPC)

Insert a temporary breakpoint at the specified virtual PC and resume all warps in the specified bitmask on a given SM.

Compared to *resumeDevice()*, this method provides finer-grain control by resuming a selected set of warps on the same SM. The main intended usage is to accelerate the single-stepping process when the target PC is known in advance. Instead of single-stepping each warp individually until the target PC is hit, the client can use this method. If an unsteppable barrier is hit by the resumed warps, this method returns early (before reaching the target PC). When this method is used, errors within CUDA kernels will no longer be reported precisely. In the situation where resuming warps is not possible, this method will return CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE. The client should then fall back to using *singleStepWarp()* or *resumeDevice()*.

Since CUDA 6.0.

See also:

resumeWarpsUntilPC

Note: DEPRECATED in CUDA 13.2: Use *resumeWarpsUntilPC* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param warpMask
[in] - the bitmask of warps to resume (1 = resume, 0 = do not resume)

Param virtPC
[in] - the virtual PC where the temporary breakpoint will be inserted

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_RUNNING_DEVICE,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE,

Return
CUDBG_ERROR_INVALID_WARP_MASK,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (*`setBreakpoint`)(uint32_t dev, uint64_t addr)

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

Before setting a breakpoint, *getAdjustedCodeAddress()* should be called to get the adjusted breakpoint address.

Since CUDA 3.2.

See also:

unsetBreakpoint

Param dev
[in] - device index

Param addr
[in] - instruction address

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***setBreakpoint31**)(uint64_t addr)

Set a breakpoint at the given instruction address.

Behaves like setBreakpoint but tries to automatically find a device for the given address.

Since CUDA 3.0.

See also:

setBreakpoint

Note: DEPRECATED in CUDA 3.2: Use *setBreakpoint* instead.

Param addr
[in] - instruction address

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***setCudaLogRules**)(const *CUDBGCudaLogRule* *rules, uint32_t numRules, uint32_t *newRulesetIndex)

Set CUDA log filtering rules.

All rules are applied in order, the first matching rule wins. If no rules match, the log message is excluded. The log level filter for each rule is applied by an equality test with the log level of the message. The API user can implement log level filtering in a flag enum style or nested levels style on its side.

The default ruleset is:

```
CUDBGCudaLogRule{CUDBG_CUDA_LOG_RULE_ACTION_SEND_ASYNC, CUDBG_CUDA_LOG_LEVEL_
↪FILTER_ANY, 0, NULL}
```

meaning send all CUDA log messages to the API client asynchronously. The default ruleset index is 0.

Since CUDA 13.4.

See also:

consumeCudaLogs

Note: This function returns immediately and the new rules are applied asynchronously in the debuggee process. It's possible that the log messages sent immediately after calling this function are not affected by the new rules.

Param rules
[in] - array of log filtering rules

Param numRules
[in] - number of log filtering rules in the array

Param newRulesetIndex
[out] - the index of the new ruleset, guaranteed to be unique and increasing, can be passed as NULL if the caller is not interested in the index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***setKernelLaunchNotificationMode**)(*CUDBGKernelLaunchNotifyMode* mode)

Set the launch notification policy.

If mode is CUDBG_KNL_LAUNCH_NOTIFY_EVENT, enable synchronous launch notification reporting (via events). This can noticeably slow down the execution of the application. If mode is CUDBG_KNL_LAUNCH_NOTIFY_DEFER, the launch notifications are not reported at all.

Since CUDA 5.5.

Param mode
[in] - mode to deliver kernel launch notifications in

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***setNotifyNewEventCallback**)(*CUDBGNotifyNewEventCallback* callback, void *userData)

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

The callback function is called for every ASYNC and SYNC event. The callback function is always called on the same thread. No API methods can be called from that thread except *getNextEvent()*, *acknowledgeSyncEvents()* (and their deprecated variants), and *consumeCudaLogs()* (since CUDA 13.4), otherwise CUDBG_ERROR_RECURSIVE_API_CALL will be returned.

Since CUDA 13.0.

See also:

acknowledgeSyncEvents

See also:

getNextEvent

See also:

consumeCudaLogs

Param callback
[in] - the callback function

Param data
[in] - a pointer to be passed to the callback when called

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

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.

Behaves like setNotifyNewEventCallback but doesn't return the host thread ID from which the event originates.

Since CUDA 3.0.

See also:

setNotifyNewEventCallback

Note: DEPRECATED in CUDA 3.2: Use *setNotifyNewEventCallback* instead.

Param callback
[in] - the callback function

Param data
[in] - a pointer to be passed to the callback when called

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***setNotifyNewEventCallback40**)(*CUDBGNotifyNewEventCallback40* callback)

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

Behaves like setNotifyNewEventCallback but doesn't allow passing in the user data pointer.

Since CUDA 3.2.

See also:

setNotifyNewEventCallback

Note: DEPRECATED in CUDA 4.1: Use *setNotifyNewEventCallback* instead.

Param callback
[in] - the callback function

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***setNotifyNewEventCallback41**)(*CUDBGNotifyNewEventCallback41* callback)

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

Behaves like setNotifyNewEventCallback but doesn't allow passing in the user data pointer. The timeout field is always 0.

Since CUDA 4.1.

See also:

setNotifyNewEventCallback

Note: DEPRECATED in CUDA 13.0: Use *setNotifyNewEventCallback* instead.

Param callback
[in] - the callback function

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***singleStepWarp**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t laneHint, uint32_t nsteps, uint32_t flags, uint64_t *warpMask)

Single step an individual warp nsteps times on a suspended CUDA device.

By default, if the warp is on a convergence barrier, resumeWarpsUntilPC is called internally to quickly advance the warp past that barrier. If the CUDBG_SINGLE_STEP_FLAGS_NO_STEP_OVER_WARP_BARRIERS flag is passed in, this optimization is not performed (which would likely lead to diverged threads becoming focused and starting to advance towards the convergence barrier). If a warp is on a block-wide barrier (or wider), other warps required to advance past the barrier are automatically resumed. The output parameter warpMask will have the warps resumed in the current SM. Warps can also be resumed in other SMs, but are not reported via the API.

This method is non-blocking if the CUDBG_SINGLE_STEP_FLAGS_NON_BLOCKING flag is passed in, otherwise it blocks until the operation is complete. The CUDBG_EVENT_SINGLE_STEP_COMPLETE event is only sent for the non-blocking mode. If a non-blocking operation is interrupted by a breakpoint or an exception, and the step is not complete, then CUDBG_EVENT_ALL_DEVICES_SUSPENDED is sent instead. NOTE: `nsteps` must be 1 if the non-blocking flag is passed in.

Since CUDA 12.4.

See also:

resumeAllDevices

See also:

resumeWarpsUntilPC

See also:

suspendAllDevices

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param laneHint
[in] - focused lane (~0 to let the API decide)

Param nsteps
[in] - number of single steps

Param flags
[in] - flags of type CUDBGSingleStepFlags to change the stepping behavior

Param warpMask
[out] - the warps that have been single-stepped

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_RUNNING_DEVICE,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE,

Return
CUDBG_ERROR_INVALID_WARP_MASK,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***singleStepWarp40**)(uint32_t dev, uint32_t sm, uint32_t wp)

Single step an individual warp on a suspended CUDA device.

Behaves like singleStepWarp41 without the output warpMask parameter.

Since CUDA 3.0.

See also:

singleStepWarp

Note: DEPRECATED in CUDA 4.1: Use *singleStepWarp* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_RUNNING_DEVICE,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE,

Return
CUDBG_ERROR_INVALID_WARP_MASK,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***singleStepWarp41**)(uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *warpMask)

Single step an individual warp on a suspended CUDA device.

Behaves like singleStepWarp65 with nsteps set to 1.

Since CUDA 4.1.

See also:

singleStepWarp

Note: DEPRECATED in CUDA 6.5: Use *singleStepWarp* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param warpMask
[out] - the warps that have been single-stepped

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_RUNNING_DEVICE,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE,

Return
CUDBG_ERROR_INVALID_WARP_MASK,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***singleStepWarp65**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t nsteps, uint64_t *warpMask)

Single step an individual warp nsteps times on a suspended CUDA device.

Behaves like singleStepWarp with no lane hint and the CUDBG_SINGLE_STEP_FLAGS_NO_STEP_OVER_WARP_BARRIERS flag set.

Since CUDA 6.5.

See also:

singleStepWarp

Note: DEPRECATED in CUDA 12.4: Use *singleStepWarp* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param nsteps
[in] - number of single steps

Param warpMask
[out] - the warps that have been single-stepped

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN_FUNCTION,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_RUNNING_DEVICE,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE,

Return
CUDBG_ERROR_INVALID_WARP_MASK,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***suspendAllDevices**)(uint32_t nonBlocking)

Suspend all running CUDA devices.

If the nonBlocking flag is non-zero, the function returns immediately and sends CUDBG_EVENT_ALL_DEVICES_SUSPENDED when the operation finishes in the background. Otherwise, if the function returns with CUDBG_SUCCESS, that guarantees that all devices have been suspended.

Since CUDA 13.2.

See also:

resumeAllDevices

Param nonBlocking
[in] - whether or not asynchronous operation is desired

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_SUSPENDED_DEVICE,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***suspendDevice**)(uint32_t dev)

Suspends a running CUDA device.

Using this method is discouraged, use *suspendAllDevices()* instead to avoid race conditions. The device has to be suspended in order to execute most operations on it. CUDBG_ERROR_SUSPENDED_DEVICE is returned if the device is already suspended.

Since CUDA 3.0.

See also:

suspendAllDevices

Param dev
[in] - device index

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_SUSPENDED_DEVICE,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***unsetBreakpoint**)(uint32_t dev, uint64_t addr)

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

Since CUDA 3.2.

See also:

setBreakpoint

Param dev
[in] - device index

Param addr
[in] - instruction address

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***unsetBreakpoint31**)(uint64_t addr)

Unset a breakpoint at the given instruction address.

Behaves like unsetBreakpoint but tries to automatically find a device for the given address.

Since CUDA 3.0.

See also:

unsetBreakpoint

Note: DEPRECATED in CUDA 3.2: Use *unsetBreakpoint* instead.

Param addr
[in] - instruction address

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

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

Write to the hardware CC register.

The CC register is no longer available in the supported hardware.

Since CUDA 6.5.

Note: DEPRECATED in CUDA 13.1: Do not use.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param val
[in] - the new value of the CC register

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writeGenericMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in any memory segment.

The address will be used to determine whether the write is to local, shared or global memory. The target address range should entirely reside within a single memory segment. Coordinate arguments are only used when relevant. They should be provided for the following segments:

- Shared memory: SM and Warp
- Local memory: SM, Warp and Lane

Since CUDA 6.0.

See also:

writeGlobalMemory

See also:

writeLocalMemory

See also:

writeParamMemory

See also:

writeSharedMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param addr
[in] - address

Param buf
[in] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (*`writeGlobalMemory`)(uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

It's not possible to access a shared memory page or an ambiguous address allocated on several devices that don't support UVA.

Since CUDA 6.0.

See also:

writeGenericMemory

See also:

writeLocalMemory

See also:

writeParamMemory

See also:

writeSharedMemory

Param addr
[in] - address

Param buf
[in] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (*`writeGlobalMemory31`)(uint32_t dev, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

This method is unsupported on Hopper and later architectures. Use newer methods: writeGlobalMemory or writeGenericMemory.

Since CUDA 3.0.

See also:

writeGlobalMemory

Note: DEPRECATED in CUDA 3.2: Use *writeGlobalMemory* instead.

Param dev
[in] - device index

Param addr
[in] - address

Param buf
[in] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***writeGlobalMemory55**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

Use newer methods: writeGlobalMemory or writeGenericMemory.

Since CUDA 3.2.

See also:

writeGlobalMemory

Note: DEPRECATED in CUDA 6.0: Use *writeGlobalMemory* instead.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param addr
[in] - address

Param buf
[in] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL,

Return
CUDBG_ERROR_NOT_SUPPORTED

CUDBGResult (***writeLocalMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in local memory.

The destination address range must be within local memory.

Since CUDA 3.0.

See also:

writeGenericMemory

See also:

writeGlobalMemory

See also:

writeParamMemory

See also:

writeSharedMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param addr
[in] - address

Param buf
[in] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_ADDRESS,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writeParamMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in param memory.

The destination address range must be within param memory.

Since CUDA 3.0.

See also:

writeGenericMemory

See also:

writeGlobalMemory

See also:

writeLocalMemory

See also:

writeSharedMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param addr
[in] - address

Param buf
[in] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writePinnedMemory**)(uint64_t addr, const void *buf, uint32_t sz)

Write to a pinned memory address.

It's not possible to access an ambiguous access allocated on several devices that don't support UVA.

Since CUDA 3.2.

See also:

readPinnedMemory

Param addr
[in] - address

Param buf
[in] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_MEMORY_MAPPING_FAILED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writePredicates**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t predicates_size, const uint32_t *predicates)

Write to hardware predicates.

This method writes to predicates_size predicates, starting from P0. Each predicate value must be either 0 or 1.

Since CUDA 6.5.

See also:

writeRegister

See also:

writeUniformPredicates

See also:

writeUniformRegister

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param predicates_size
[in] - predicates count

Param predicates
[in] - predicate values

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writeRegister**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t regno, uint32_t val)

Write to a hardware register.

Since CUDA 3.0.

See also:

writePredicates

See also:

writeUniformPredicates

See also:

writeUniformRegister

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param regno
[in] - register number

Param val
[in] - value

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writeRpcRegisters**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, const uint32_t *rpcLo, const uint32_t *rpcHi)

Write the RPC.LO and/or RPC.HI hardware registers for a specific active thread.

Only active (non-diverged) threads may be written. This means that it's impossible to use this method to change the PC of a diverged thread, but also that it's impossible to change the PC of an active thread with it since the active thread does not restore its PC from the RPC registers.

Passing NULL for `rpcLo` or `rpcHi` causes that register to be skipped. Passing NULL for both is an error.

Since CUDA 13.4.

See also:

readRpcRegisters

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param ln
[in] - lane index

Param rpcLo
[in] - new value for RPC.LO (lower 32 bits), or NULL to skip

Param rpcHi
[in] - new value for RPC.HI (upper 32 bits), or NULL to skip

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INVALID_DEVICE,

Return
CUDBG_ERROR_INVALID_SM,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_LANE,

Return
CUDBG_ERROR_NOT_SUPPORTED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writeSharedMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in shared memory.

The destination address range must be within shared memory.

Since CUDA 3.0.

See also:

writeGenericMemory

See also:

writeGlobalMemory

See also:

writeLocalMemory

See also:

writeParamMemory

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param addr
[in] - address

Param buf
[in] - buffer

Param sz
[in] - buffer size in bytes

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writeTcgen05TmemMemory**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t addr, const void *buf, uint32_t sz)

Write tcgen05 Tensor Memory data for the CTA containing a warp.

Since CUDA 13.5.

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param addr
[in] - Tensor Memory address: bits 31:16 are lane, bits 15:0 are column

Param buf
[in] - buffer containing Tensor Memory data

Param sz
[in] - number of bytes to write

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INVALID_MEMORY_ACCESS,

Return
CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM,

Return
CUDBG_ERROR_NOT_SUPPORTED,

Return
CUDBG_ERROR_UNKNOWN,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writeUniformPredicates**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t predicates_size, const uint32_t *predicates)

Write to hardware uniform predicates.

This method writes to predicates_size uniform predicates, starting from UP0. Each predicate value must be either 0 or 1.

Since CUDA 10.0.

See also:

writePredicates

See also:

writeRegister

See also:

writeUniformRegister

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param predicates_size
[in] - predicates count

Param predicates
[in] - predicate values

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

CUDBGResult (***writeUniformRegister**)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t regno, uint32_t val)

Write to a hardware uniform register.

Since CUDA 10.0.

See also:

writePredicates

See also:

writeRegister

See also:

writeUniformPredicates

Param dev
[in] - device index

Param sm
[in] - SM index

Param wp
[in] - warp index

Param regno
[in] - register number

Param val
[in] - value

Return
CUDBG_SUCCESS,

Return
CUDBG_ERROR_INVALID_ARGS,

Return
CUDBG_ERROR_UNINITIALIZED,

Return
CUDBG_ERROR_INTERNAL,

Return
CUDBG_ERROR_INVALID_WARP,

Return
CUDBG_ERROR_INITIALIZATION_FAILURE,

Return
CUDBG_ERROR_INVALID_CONTEXT,

Return
CUDBG_ERROR_RECURSIVE_API_CALL

3.2. CUDBGAttributeValuePair

struct **CUDBGAttributeValuePair**

Grid attribute-value pair.

Public Members

CUDBGAttribute **attribute**

The attribute to query.

uint64_t **value**

The value of the attribute.

3.3. CUDBGCbuWarpState

struct **CUDBGCbuWarpState**

Warp state in the CBU (Convergence Barrier Unit).

Public Members

uint32_t **activeMask**

Active threads mask.

uint32_t **barrierMasks**[16]

Convergence barrier participation masks.

uint32_t **collectiveMask**

Mask of threads that are part of a collective operation.

uint32_t **exitedMask**

Exited threads mask.

CUDBGCbuThreadState **threadState**[32]

Thread state for each warp lane.

3.4. CUDBGCudaLogMessage

struct **CUDBGCudaLogMessage**

CUDA Log Message.

Public Members

CUDBGCudaLogLevel **logLevel**

The log severity level of the message.

uint32_t **matchedRuleIndex**

The rule index of the rule that matched the log message.

uint32_t **matchedRulesetIndex**

The index of the ruleset that matched the log message.

char **message**[256]

The log message string.

uint64_t **osThreadId**

The native OS thread ID of the thread that generated the log message.

uint64_t **unixTimestampNs**

The timestamp the log message was received at, in nanoseconds since the Unix epoch.

3.5. CUDBGCudaLogMessage129

struct **CUDBGCudaLogMessage129**

CUDA Log Message for API version 12.9.

See also:

CUDBGCudaLogMessage

Note: DEPRECATED: Use *CUDBGCudaLogMessage* instead. For documentation of individual fields, see the documentation of the *CUDBGCudaLogMessage* struct instead.

Public Members

CUDBGCudaLogLevel **logLevel**

The log severity level of the message.

char **message**[256]

The log message string.

uint32_t **osThreadId**

The OS thread ID of the thread that generated the log message.

Note: This is a *possibly truncated* native OS thread ID. Use *CUDBGCudaLogMessage* instead for the full value.

uint64_t **unixTimestampNs**

The timestamp the log message was received at, in nanoseconds since the Unix epoch.

3.6. CUDBGCudaLogRule

struct **CUDBGCudaLogRule**

CUDA Log filtering rule.

Public Members

CUDBGCudaLogRuleAction **action**

Action to take for the log message.

CUDBGCudaLogLevelFilter **logLevelFilter**

A particular log level (or any)

const char ***messageFilterRegex**

A particular message content or any if NULL.

Note: This field accepts C++ modified ECMAScript regex syntax (the std::regex default)

uint64_t **osThreadIdFilter**

A particular OS thread ID or for any thread if 0.

The user should only set this to TID values seen via consumeCudaLogs().

3.7. CUDBGDeviceInfo

struct **CUDBGDeviceInfo**

Device-level information.

This is the first element in the deviceInfoBuffer, and is always present. getDeviceInfo() takes a deviceId as input, so no need to explicitly pass it back here. Only "valid & updated" SMs/Warps/Lanes are included in the buffer, which allows us to determine indexes without having to encode an explicit ID field in the following buffer datastructures.

Public Members

uint32_t **deviceAttributeFlags**

Bitmask of CUDBGDeviceInfoAttribute_t enums for a device.

CUDBGDeviceInfoQueryType_t **responseType**

Response type.

3.8. CUDBGDeviceInfoSizes

struct **CUDBGDeviceInfoSizes**

Sizes of the various structs returned by the batched device update APIs.

No explicit version field - implied by debugAPI major.minor.revision.

Public Members

uint32_t **deviceInfoAttributeSizes**[32]

Device info attribute sizes.

uint32_t **deviceInfoSize**

Device info size.

uint32_t **laneInfoAttributeSizes**[32]

Lane (thread) info attribute sizes.

uint32_t **laneInfoSize**

Lane (thread) info size.

uint32_t **requiredBufferSize**

Required buffer size.

uint32_t **smInfoAttributeSizes**[32]
SM info attribute sizes.

uint32_t **smInfoSize**
SM info size.

uint32_t **warpInfoAttributeSizes**[32]
Warp info attribute sizes.

uint32_t **warpInfoSize**
Warp info size.

3.9. CUDBGEvent

Unions

cases_st
Information for each type of event.

struct **CUDBGEvent**
Event information container.

Public Members

union *CUDBGEvent*::*cases_st* **cases**

CUDBGEventKind **kind**

union **cases_st**
Information for each type of event.

Public Members

struct *CUDBGEvent*::*cases_st*::*allDevicesSuspended_st* **allDevicesSuspended**

struct *CUDBGEvent*::*cases_st*::*contextCreate_st* **contextCreate**

struct *CUDBGEvent*::*cases_st*::*contextDestroy_st* **contextDestroy**

struct *CUDBGEvent*::*cases_st*::*contextPop_st* **contextPop**

struct *CUDBGEvent*::*cases_st*::*contextPush_st* **contextPush**

struct *CUDBGEvent*::*cases_st*::*cudaLogsRulesetChanged_st* **cudaLogsRulesetChanged**

struct *CUDBGEvent*::*cases_st*::*elfImageLoaded_st* **elfImageLoaded**

struct *CUDBGEvent*::*cases_st*::*elfImageUnloaded_st* **elfImageUnloaded**

struct *CUDBGEvent*::*cases_st*::*functionsLoaded_st* **functionsLoaded**

struct *CUDBGEvent*::*cases_st*::*internalError_st* **internalError**

struct *CUDBGEvent*::*cases_st*::*kernelFinished_st* **kernelFinished**

struct *CUDBGEvent*::*cases_st*::*kernelReady_st* **kernelReady**

struct *CUDBGEvent*::*cases_st*::*singleStepComplete_st* **singleStepComplete**

struct **allDevicesSuspended_st**

Information about an event which forced all devices to be suspended.

Public Members

uint64_t **brokenDevicesMask**

Device bitmask.

This mask has bits set for devices with any warps that hit a breakpoint.

uint64_t **faultedDevicesMask**

Device bitmask.

This mask has bits set for devices with any warps that hit an exception.

struct **contextCreate_st**

Information about the context being created.

Public Members

uint64_t **context**

Context handle of the context being created.

uint32_t **dev**

Device index of the context.

uint32_t **tid**
Host thread id (or LWP id) of the thread hosting the context (Linux only).

struct **contextDestroy_st**
Information about the context being destroyed.

Public Members

uint64_t **context**
Context handle of the context being destroyed.

uint32_t **dev**
Device index of the context.

uint32_t **tid**
Host thread id (or LWP id) of the thread hosting the context (Linux only).

struct **contextPop_st**
Information about the context being popped.

Public Members

uint64_t **context**
Context handle of the context being popped.

uint32_t **dev**
Device index of the context.

uint32_t **tid**
Host thread id (or LWP id) of the thread hosting the context (Linux only).

struct **contextPush_st**
Information about the context being pushed.

Public Members

uint64_t **context**
Context handle of the context being pushed.

uint32_t **dev**
Device index of the context.

uint32_t **tid**

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

struct **cudaLogsRulesetChanged_st**

Information about the CUDA log ruleset being changed.

Public Members

uint32_t **rulesetIndex**

The index of the newly applied ruleset.

struct **elfImageLoaded_st**

Information about the loaded ELF image.

Public Members

uint64_t **context**

Context handle of the loaded module.

uint32_t **dev**

Device index of the loaded module.

uint64_t **handle**

ELF image handle.

Note: Since CUDA 13.5, if CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LAZY_FUNCTION_FINA is enabled, then this field no longer has the same value as the module field. There may be multiple ELF images associated with a single CUDA module.

---

uint64_t **module**

Loaded module handle.

uint32_t **properties**

ELF image properties.

uint64_t **size**

Size of the ELF image (64-bit).

struct **elfImageUnloaded_st**

Information about the ELF image about to be unloaded.

Public Members

uint64_t **context**

Context handle of the module being unloaded.

uint32_t **dev**

Device index of the module being unloaded.

uint64_t **handle**

ELF image handle.

Note: Since CUDA 13.5, if CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LAZY_FUNCTION_FINA is enabled, then this field no longer has the same value as the module field. There may be multiple ELF images associated with a single CUDA module.

---

uint64_t **module**

Module handle of the module being unloaded.

uint64_t **size**

Size of the ELF image (64-bit).

struct **functionsLoaded_st**

Information about the functions being lazily loaded.

Public Members

uint64_t **context**

Context handle of the module containing the functions.

uint32_t **count**

Functions count.

uint32_t **dev**

Device index of the module containing the functions.

uint64_t **elfImageHandle**

Handle of the ELF image containing the functions.

Note: This field value will match the handle value field from the elfImageLoaded event.

---

struct **internalError_st**

Information about internal erros.

Public Members

CUDBGResult **errorType**
Type of the internal error.

struct **kernelFinished_st**
Information about the kernel that just terminated.

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**
Context handle of the kernel.

uint32_t **dev**
Device index of the kernel.

uint64_t **function**
Function handle of the kernel.

uint64_t **functionEntry**
Entry address of the kernel.

uint64_t **gridId**
Grid ID of the kernel.

uint64_t **module**
Module handle of the kernel.

uint32_t **tid**
Host thread id (or LWP id) of the thread hosting the kernel (Linux only).

struct **kernelReady_st**
Information about the kernel ready to be launched.

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

CuDim3 **blockDim**
Block dimensions of the kernel.

uint64_t **context**
Context handle of the kernel.

uint32_t **dev**
Device index of the kernel.

uint64_t **function**
Function handle of the kernel.

uint64_t **functionEntry**
Entry address of the kernel.

CuDim3 **gridDim**
Grid dimensions of the kernel.

uint64_t **gridId**
Grid ID of the kernel.

uint64_t **module**
Module handleof the kernel.

CUDBGKernelOrigin **origin**
Origin of the kernel (CPU or GPU).

uint64_t **reserved0**
Reserved deprecated field kept for ABI compatibility.

Note: DEPRECATED: Since CUDA 13.4, this field is no longer used. Used to be ID of the parent grid (in case of a device-launched CDP grid).

uint32_t **tid**
Host thread id (or LWP id) of the thread hosting the kernel (Linux only).

CUDBGKernelType **type**
Type of the kernel: system or application.

struct **singleStepComplete_st**
Information about a single step operation that has completed.

Public Members

uint64_t **brokenDevicesMask**

Device bitmask.

This mask has bits set for devices with any warps that hit a breakpoint during the step.

uint64_t **context**

Context that was stepping.

uint32_t **dev**

Device that was stepping.

uint64_t **faultedDevicesMask**

Device bitmask.

This mask has bits set for devices with any warps that hit an exception during the step.

uint64_t **finalWarpMask**

Warps that ended up being stepped (only contains warps from the original SM, other SMs could be stepped and are not reported)

uint64_t **originalWarpMask**

Warps that were requested to be stepped.

uint32_t **sm**

SM that was stepping.

CUDBGSingleStepType **type**

Which single step method has initiated stepping.

3.10. CUDBGEvent30

Unions

cases30_st

struct **CUDBGEvent30**

Event information container for API version 3.0.

See also:

CUDBGEvent

Note: DEPRECATED: Use *CUDBGEvent* instead. For documentation of individual fields, see the documentation of the *CUDBGEvent* struct instead.

Public Members

union *CUDBGEvent30*::*cases30_st* **cases**

CUDBGEventKind **kind**

union **cases30_st**

Public Members

struct *CUDBGEvent30*::*cases30_st*::*elfImageLoaded30_st* **elfImageLoaded**

struct *CUDBGEvent30*::*cases30_st*::*kernelFinished30_st* **kernelFinished**

struct *CUDBGEvent30*::*cases30_st*::*kernelReady30_st* **kernelReady**

struct **elfImageLoaded30_st**

Public Members

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint32_t **size**

struct **kernelFinished30_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint32_t **dev**

uint32_t **gridId**

uint32_t **tid**

struct **kernelReady30_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint32_t **dev**

uint32_t **gridId**

uint32_t **tid**

3.11. CUDBGEvent30::cases30_st::elfImageLoaded30_st

struct **elfImageLoaded30_st**

Public Members

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint32_t **size**

3.12. CUDBGEvent30::cases30_st::kernelFinished30_st

struct **kernelFinished30_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint32_t **dev**

uint32_t **gridId**

uint32_t **tid**

3.13. CUDBGEvent30::cases30_st::kernelReady30_st

struct **kernelReady30_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint32_t **dev**

uint32_t **gridId**

uint32_t **tid**

3.14. CUDBGEvent32

Unions

cases32_st

struct **CUDBGEvent32**

Event information container for API version 3.2.

See also:

CUDBGEvent

Note: DEPRECATED: Use *CUDBGEvent* instead. For documentation of individual fields, see the documentation of the *CUDBGEvent* struct instead.

Public Members

union *CUDBGEvent32*::*cases32_st* **cases**

CUDBGEventKind **kind**

union **cases32_st**

Public Members

struct *CUDBGEvent32*::*cases32_st*::*contextCreate32_st* **contextCreate**

struct *CUDBGEvent32*::*cases32_st*::*contextDestroy32_st* **contextDestroy**

struct *CUDBGEvent32*::*cases32_st*::*contextPop32_st* **contextPop**

struct *CUDBGEvent32*::*cases32_st*::*contextPush32_st* **contextPush**

struct *CUDBGEvent32*::*cases32_st*::*elfImageLoaded32_st* **elfImageLoaded**

struct *CUDBGEvent32*::*cases32_st*::*kernelFinished32_st* **kernelFinished**

struct *CUDBGEvent32*::*cases32_st*::*kernelReady32_st* **kernelReady**

struct **contextCreate32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextDestroy32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextPop32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextPush32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **elfImageLoaded32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **module**

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint32_t **size**

struct **kernelFinished32_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

struct **kernelReady32_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

3.15. CUDBGEvent32::cases32_st::contextCreate32_st

struct **contextCreate32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.16. CUDBGEvent32::cases32_st::contextDestroy32_st

struct **contextDestroy32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.17. CUDBGEvent32::cases32_st::contextPop32_st

struct **contextPop32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.18. CUDBGEvent32::cases32_st::contextPush32_st

struct **contextPush32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.19. CUDBGEvent32::cases32_st::elfImageLoaded32_st

struct **elfImageLoaded32_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **module**

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint32_t **size**

3.20. CUDBGEvent32::cases32_st::kernelFinished32_st

struct **kernelFinished32_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

3.21. CUDBGEvent32::cases32_st::kernelReady32_st

struct **kernelReady32_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

3.22. CUDBGEvent42

Unions

cases42_st

struct **CUDBGEvent42**

Event information container for API version 4.2.

See also:

CUDBGEvent

Note: DEPRECATED: Use *CUDBGEvent* instead. For documentation of individual fields, see the documentation of the *CUDBGEvent* struct instead.

Public Members

union *CUDBGEvent42*::*cases42_st* **cases**

CUDBGEventKind **kind**

union **cases42_st**

Public Members

struct *CUDBGEvent42*::*cases42_st*::*contextCreate42_st* **contextCreate**

struct *CUDBGEvent42*::*cases42_st*::*contextDestroy42_st* **contextDestroy**

struct *CUDBGEvent42*::*cases42_st*::*contextPop42_st* **contextPop**

struct *CUDBGEvent42*::*cases42_st*::*contextPush42_st* **contextPush**

struct *CUDBGEvent42*::*cases42_st*::*elfImageLoaded42_st* **elfImageLoaded**

struct *CUDBGEvent42*::*cases42_st*::*kernelFinished42_st* **kernelFinished**

struct *CUDBGEvent42*::*cases42_st*::*kernelReady42_st* **kernelReady**

struct **contextCreate42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextDestroy42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextPop42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextPush42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **elfImageLoaded42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **module**

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint64_t **size**

uint32_t **size32**

struct **kernelFinished42_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

struct **kernelReady42_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

CuDim3 **blockDim**

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

CuDim3 **gridDim**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

CUDBGKernelType **type**

3.23. CUDBGEvent42::cases42_st::contextCreate42_st

struct **contextCreate42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.24. CUDBGEvent42::cases42_st::contextDestroy42_st

struct **contextDestroy42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.25. CUDBGEvent42::cases42_st::contextPop42_st

struct **contextPop42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.26. CUDBGEvent42::cases42_st::contextPush42_st

struct **contextPush42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.27. CUDBGEvent42::cases42_st::elfImageLoaded42_st

struct **elfImageLoaded42_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **module**

char *nonRelocatedElfImage

char *relocatedElfImage

uint64_t size

uint32_t size32

3.28. CUDBGEvent42::cases42_st::kernelFinished42_st

struct kernelFinished42_st

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t context

uint32_t dev

uint64_t function

uint64_t functionEntry

uint32_t gridId

uint64_t module

uint32_t tid

3.29. CUDBGEvent42::cases42_st::kernelReady42_st

struct kernelReady42_st

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

CuDim3 **blockDim**

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

CuDim3 **gridDim**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

CUDBGKernelType **type**

3.30. CUDBGEvent50

Unions

cases50_st

struct **CUDBGEvent50**

Event information container for API version 5.0.

See also:

CUDBGEvent

Note: DEPRECATED: Use *CUDBGEvent* instead. For documentation of individual fields, see the documentation of the *CUDBGEvent* struct instead.

Public Members

union *CUDBGEvent50*::*cases50_st* **cases**

CUDBGEventKind **kind**

union **cases50_st**

Public Members

struct *CUDBGEvent50*::*cases50_st*::*contextCreate50_st* **contextCreate**

struct *CUDBGEvent50*::*cases50_st*::*contextDestroy50_st* **contextDestroy**

struct *CUDBGEvent50*::*cases50_st*::*contextPop50_st* **contextPop**

struct *CUDBGEvent50*::*cases50_st*::*contextPush50_st* **contextPush**

struct *CUDBGEvent50*::*cases50_st*::*elfImageLoaded50_st* **elfImageLoaded**

struct *CUDBGEvent50*::*cases50_st*::*internalError50_st* **internalError**

struct *CUDBGEvent50*::*cases50_st*::*kernelFinished50_st* **kernelFinished**

struct *CUDBGEvent50*::*cases50_st*::*kernelReady50_st* **kernelReady**

struct **contextCreate50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextDestroy50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextPop50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextPush50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **elfImageLoaded50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **module**

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint64_t **size**

uint32_t **size32**

struct **internalError50_st**

Public Members

CUDBGResult **errorType**

struct **kernelFinished50_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

struct **kernelReady50_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

CuDim3 **blockDim**

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

CuDim3 **gridDim**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

CUDBGKernelType **type**

3.31. CUDBGEvent50::cases50_st::contextCreate50_st

struct **contextCreate50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.32. CUDBGEvent50::cases50_st::contextDestroy50_st

struct **contextDestroy50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.33. CUDBGEvent50::cases50_st::contextPop50_st

struct **contextPop50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.34. CUDBGEvent50::cases50_st::contextPush50_st

struct **contextPush50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.35. CUDBGEvent50::cases50_st::elfImageLoaded50_st

struct **elfImageLoaded50_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **module**

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint64_t **size**

uint32_t **size32**

3.36. CUDBGEvent50::cases50_st::internalError50_st

struct **internalError50_st**

Public Members

CUDBGResult **errorType**

3.37. CUDBGEvent50::cases50_st::kernelFinished50_st

struct **kernelFinished50_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

3.38. CUDBGEvent50::cases50_st::kernelReady50_st

struct **kernelReady50_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

CuDim3 **blockDim**

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

CuDim3 **gridDim**

uint32_t **gridId**

uint64_t **module**

uint32_t **tid**

CUDBGKernelType **type**

3.39. CUDBGEvent55

Unions

cases55_st

struct **CUDBGEvent55**

Event information container for API version 5.5.

See also:

CUDBGEvent

Note: DEPRECATED: Use *CUDBGEvent* instead. For documentation of individual fields, see the documentation of the *CUDBGEvent* struct instead.

Public Members

union *CUDBGEvent55*::*cases55_st* **cases**

CUDBGEventKind **kind**

union **cases55_st**

Public Members

struct *CUDBGEvent55*::*cases55_st*::*contextCreate55_st* **contextCreate**

struct *CUDBGEvent55*::*cases55_st*::*contextDestroy55_st* **contextDestroy**

struct *CUDBGEvent55*::*cases55_st*::*contextPop55_st* **contextPop**

struct *CUDBGEvent55*::*cases55_st*::*contextPush55_st* **contextPush**

struct *CUDBGEvent55*::*cases55_st*::*elfImageLoaded55_st* **elfImageLoaded**

struct *CUDBGEvent55*::*cases55_st*::*internalError55_st* **internalError**

struct *CUDBGEvent55*::*cases55_st*::*kernelFinished55_st* **kernelFinished**

struct *CUDBGEvent55*::*cases55_st*::*kernelReady55_st* **kernelReady**

struct **contextCreate55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextDestroy55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextPop55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **contextPush55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

struct **elfImageLoaded55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **module**

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint64_t **size**

uint32_t **size32**

struct **internalError55_st**

Public Members

CUDBGResult **errorType**

struct **kernelFinished55_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **gridId64**

uint64_t **module**

uint32_t **tid**

struct **kernelReady55_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

CuDim3 **blockDim**

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

CuDim3 **gridDim**

uint32_t **gridId**

uint64_t **gridId64**

uint64_t **module**

CUDBGKernelOrigin **origin**

uint64_t **reserved0**

Reserved deprecated field kept for ABI compatibility.

Note: DEPRECATED: Since CUDA 13.4, this field is no longer used. Used to be ID of the parent grid (in case of a device-launched CDP grid).

uint32_t **tid**

CUDBGKernelType **type**

3.40. CUDBGEvent55::cases55_st::contextCreate55_st

struct **contextCreate55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.41. CUDBGEvent55::cases55_st::contextDestroy55_st

struct **contextDestroy55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.42. CUDBGEvent55::cases55_st::contextPop55_st

struct **contextPop55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.43. CUDBGEvent55::cases55_st::contextPush55_st

struct **contextPush55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint32_t **tid**

3.44. CUDBGEvent55::cases55_st::elfImageLoaded55_st

struct **elfImageLoaded55_st**

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **module**

char ***nonRelocatedElfImage**

char ***relocatedElfImage**

uint64_t **size**

uint32_t **size32**

3.45. CUDBGEvent55::cases55_st::internalError55_st

struct **internalError55_st**

Public Members

CUDBGResult **errorType**

3.46. CUDBGEvent55::cases55_st::kernelFinished55_st

struct **kernelFinished55_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

uint32_t **gridId**

uint64_t **gridId64**

uint64_t **module**

uint32_t **tid**

3.47. CUDBGEvent55::cases55_st::kernelReady55_st

struct **kernelReady55_st**

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

CuDim3 **blockDim**

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

CuDim3 **gridDim**

uint32_t **gridId**

uint64_t **gridId64**

uint64_t **module**

CUDBGKernelOrigin **origin**

uint64_t **reserved0**

Reserved deprecated field kept for ABI compatibility.

Note: DEPRECATED: Since CUDA 13.4, this field is no longer used. Used to be ID of the parent grid (in case of a device-launched CDP grid).

uint32_t **tid**

CUDBGKernelType **type**

3.48. CUDBGEvent::cases_st::allDevicesSuspended_st

struct **allDevicesSuspended_st**

Information about an event which forced all devices to be suspended.

Public Members

uint64_t **brokenDevicesMask**

Device bitmask.

This mask has bits set for devices with any warps that hit a breakpoint.

uint64_t **faultedDevicesMask**

Device bitmask.

This mask has bits set for devices with any warps that hit an exception.

3.49. CUDBGEvent::cases_st::contextCreate_st

struct **contextCreate_st**

Information about the context being created.

Public Members

uint64_t **context**

Context handle of the context being created.

uint32_t **dev**

Device index of the context.

uint32_t **tid**

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

3.50. CUDBGEvent::cases_st::contextDestroy_st

struct **contextDestroy_st**

Information about the context being destroyed.

Public Members

uint64_t **context**

Context handle of the context being destroyed.

uint32_t **dev**

Device index of the context.

uint32_t **tid**

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

3.51. CUDBGEvent::cases_st::contextPop_st

struct **contextPop_st**

Information about the context being popped.

Public Members

uint64_t **context**

Context handle of the context being popped.

uint32_t **dev**

Device index of the context.

uint32_t **tid**

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

3.52. CUDBGEvent::cases_st::contextPush_st

struct **contextPush_st**

Information about the context being pushed.

Public Members

uint64_t **context**

Context handle of the context being pushed.

uint32_t **dev**

Device index of the context.

uint32_t **tid**

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

3.53. CUDBGEvent::cases_st::cudaLogsRulesetChanged_

struct **cudaLogsRulesetChanged_st**

Information about the CUDA log ruleset being changed.

Public Members

uint32_t **rulesetIndex**

The index of the newly applied ruleset.

3.54. CUDBGEvent::cases_st::elfImageLoaded_st

struct **elfImageLoaded_st**

Information about the loaded ELF image.

Public Members

uint64_t **context**

Context handle of the loaded module.

uint32_t **dev**

Device index of the loaded module.

uint64_t **handle**

ELF image handle.

Note: Since CUDA 13.5, if CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LAZY_FUNCTION_FINALIZATIO is enabled, then this field no longer has the same value as the module field. There may be multiple ELF images associated with a single CUDA module.

---

uint64_t **module**

Loaded module handle.

uint32_t **properties**

ELF image properties.

uint64_t **size**

Size of the ELF image (64-bit).

3.55. CUDBGEvent::cases_st::elfImageUnloaded_st

struct **elfImageUnloaded_st**

Information about the ELF image about to be unloaded.

Public Members

uint64_t **context**

Context handle of the module being unloaded.

uint32_t **dev**

Device index of the module being unloaded.

uint64_t **handle**

ELF image handle.

Note: Since CUDA 13.5, if CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LAZY_FUNCTION_FINALIZATIO is enabled, then this field no longer has the same value as the module field. There may be multiple ELF images associated with a single CUDA module.

---

uint64_t **module**

Module handle of the module being unloaded.

uint64_t **size**

Size of the ELF image (64-bit).

3.56. CUDBGEvent::cases_st::functionsLoaded_st

struct **functionsLoaded_st**

Information about the functions being lazily loaded.

Public Members

uint64_t **context**

Context handle of the module containing the functions.

uint32_t **count**

Functions count.

uint32_t **dev**
Device index of the module containing the functions.

uint64_t **elfImageHandle**
Handle of the ELF image containing the functions.

Note: This field value will match the handle value field from the elfImageLoaded event.

3.57. CUDBGEvent::cases_st::internalError_st

struct **internalError_st**
Information about internal erros.

Public Members

CUDBGResult **errorType**
Type of the internal error.

3.58. CUDBGEvent::cases_st::kernelFinished_st

struct **kernelFinished_st**
Information about the kernel that just terminated.

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

uint64_t **context**
Context handle of the kernel.

uint32_t **dev**
Device index of the kernel.

uint64_t **function**
Function handle of the kernel.

uint64_t **functionEntry**
: Entry address of the kernel.

uint64_t **gridId**
: Grid ID of the kernel.

uint64_t **module**
: Module handle of the kernel.

uint32_t **tid**
: Host thread id (or LWP id) of the thread hosting the kernel (Linux only).

3.59. CUDBGEvent::cases_st::kernelReady_st

struct **kernelReady_st**
: Information about the kernel ready to be launched.

Note: DEPRECATED: This event is unreliable and will be removed in a future release.

Public Members

CuDim3 **blockDim**
: Block dimensions of the kernel.

uint64_t **context**
: Context handle of the kernel.

uint32_t **dev**
: Device index of the kernel.

uint64_t **function**
: Function handle of the kernel.

uint64_t **functionEntry**
: Entry address of the kernel.

CuDim3 **gridDim**
: Grid dimensions of the kernel.

uint64_t **gridId**
: Grid ID of the kernel.

uint64_t **module**

Module handleof the kernel.

CUDBGKernelOrigin **origin**

Origin of the kernel (CPU or GPU).

uint64_t **reserved0**

Reserved deprecated field kept for ABI compatibility.

Note: DEPRECATED: Since CUDA 13.4, this field is no longer used. Used to be ID of the parent grid (in case of a device-launched CDP grid).

uint32_t **tid**

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

CUDBGKernelType **type**

Type of the kernel: system or application.

3.60. CUDBGEvent::cases_st::singleStepComplete_st

struct **singleStepComplete_st**

Information about a single step operation that has completed.

Public Members

uint64_t **brokenDevicesMask**

Device bitmask.

This mask has bits set for devices with any warps that hit a breakpoint during the step.

uint64_t **context**

Context that was stepping.

uint32_t **dev**

Device that was stepping.

uint64_t **faultedDevicesMask**

Device bitmask.

This mask has bits set for devices with any warps that hit an exception during the step.

uint64_t **finalWarpMask**

Warps that ended up being stepped (only contains warps from the original SM, other SMs could be stepped and are not reported)

uint64_t **originalWarpMask**

Warps that were requested to be stepped.

uint32_t **sm**

SM that was stepping.

CUDBGSingleStepType **type**

Which single step method has initiated stepping.

3.61. CUDBGEventCallbackData

struct **CUDBGEventCallbackData**

Callback data passed to callback set with setNotifyNewEventCallback function.

Public Members

uint32_t **tid**

Host thread id of the context generating the event.

Zero if not available.

void ***userData**

User data passed to the callback.

3.62. CUDBGEventCallbackData40

struct **CUDBGEventCallbackData40**

Callback data passed to callback set with setNotifyNewEventCallback40 function.

Note: DEPRECATED: Use *CUDBGEventCallbackData* instead.

Public Members

uint32_t **tid**

Host thread id of the context generating the event.

Zero if not available.

3.63. CUDBGEventCallbackData41

struct **CUDBGEventCallbackData41**

Callback data passed to callback set with setNotifyNewEventCallback41 function.

Note: DEPRECATED: Use *CUDBGEventCallbackData* instead.

Public Members

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 reponse from the debugger to a previous event.

It is up to the debugger to decide what to do in response to a timeout.

3.64. CUDBGGridInfo

struct **CUDBGGridInfo**

Information about a CUDA grid.

Public Members

CuDim3 **blockDim**

Block dimensions.

CuDim3 **clusterDim**

Number of blocks in the cluster.

uint64_t **context**

Context handle of the context this grid belongs to.

uint32_t **dev**

Index of the device this grid is running on.

uint64_t **elfImageHandle**

Handle of the ELF image this grid's entry function belongs to.

Note: This field will match the handle value field from the elfImageLoaded event.

uint64_t **function**

Function handle of the function corresponding to this grid.

uint64_t **functionEntry**

Entry address of the function corresponding to this grid.

CuDim3 **gridDim**

Grid dimensions.

uint64_t **gridId64**

Grid ID of this grid.

CUDBGKernelOrigin **origin**

Origin of this grid: CPU or GPU.

CuDim3 **preferredClusterDim**

Preferred number of blocks in the cluster.

uint64_t **reserved0**

Reserved deprecated field kept for ABI compatibility.

Note: DEPRECATED: Since CUDA 13.4, this field is no longer used. Used to be ID of the parent grid (in case of a device-launched CDP grid).

uint32_t **tid**

Thread ID of the host thread that launched this grid.

CUDBGKernelType **type**

Grid type: system or application.

3.65. CUDBGGridInfo120

struct **CUDBGGridInfo120**

Grid info.

See also:

CUDBGGridInfo

Note: DEPRECATED: Use *CUDBGGridInfo* instead. For documentation of individual fields, see the documentation of the *CUDBGGridInfo* struct instead.

Public Members

CuDim3 **blockDim**

CuDim3 **clusterDim**

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

CuDim3 **gridDim**

uint64_t **gridId64**

uint64_t **module**

CUDBGKernelOrigin **origin**

uint64_t **reserved0**

Reserved deprecated field kept for ABI compatibility.

Note: DEPRECATED: Since CUDA 13.4, this field is no longer used. Used to be ID of the parent grid (in case of a device-launched CDP grid).

uint32_t **tid**

CUDBGKernelType **type**

3.66. CUDBGGridInfo55

struct **CUDBGGridInfo55**

Grid info.

See also:

CUDBGGridInfo

Note: DEPRECATED: Use *CUDBGGridInfo* instead. For documentation of individual fields, see the documentation of the *CUDBGGridInfo* struct instead.

Public Members

CuDim3 **blockDim**

uint64_t **context**

uint32_t **dev**

uint64_t **function**

uint64_t **functionEntry**

CuDim3 **gridDim**

uint64_t **gridId64**

uint64_t **module**

CUDBGKernelOrigin **origin**

uint64_t **reserved0**

Reserved deprecated field kept for ABI compatibility.

Note: DEPRECATED: Since CUDA 13.4, this field is no longer used. Used to be ID of the parent grid (in case of a device-launched CDP grid).

uint32_t **tid**

CUDBGKernelType **type**

3.67. CUDBGLaneInfo

struct **CUDBGLaneInfo**

Public Members

uint64_t **virtualPC**

(VA) PC of the thread.

3.68. CUDBGLaneState

struct **CUDBGLaneState**

Lane state (state of a single thread).

Public Members

CUDBGException_t **exception**

Exception of the thread (if any).

CuDim3 **threadIdx**

Thread index of the thread.

uint64_t **virtualPC**

(VA) PC of the thread.

3.69. CUDBGLoadedFunctionInfo

struct **CUDBGLoadedFunctionInfo**

Information about a lazily loaded function.

Public Members

uint64_t **address**

Address of a loaded function.

uint64_t **sectionIndex**

Section index of a loaded function.

3.70. CUDBGMemoryInfo

struct **CUDBGMemoryInfo**

Memory information.

Public Members

uint64_t **size**

Size of the memory region.

uint64_t **startAddress**

Start address of the memory region.

3.71. CUDBGSMInfo

struct **CUDBGSMInfo**

SM-level information.

Public Members

uint32_t **smAttributeFlags**
Bitmask of CUDBGSmInfoAttribute_t enums for a SM.

uint64_t **warpBrokenMask**
Broken warps mask.

uint64_t **warpValidMask**
Valid warps mask.

3.72. CUDBGTcgen05TmemInfo

struct **CUDBGTcgen05TmemInfo**
tcgen05 Tensor Memory information for a warp's CTA.

Public Members

uint32_t **allocationMask**
Allocated tensor memory column chunks for this CTA.

uint32_t **allocationPhase**
Tensor Memory allocation phase.
See CUDBGTcgen05TmemAllocationPhase.

uint32_t **allocationStartMask**
Allocation-start tensor memory column chunks for this CTA.

uint32_t **ctaGroup**
CTA group value.
Valid values are 1 or 2.

3.73. CUDBGWarpInfo

struct **CUDBGWarpInfo**
Warp-level information.

Public Members

uint32_t **activeLanes**

Active lanes (threads) mask.

CuDim3 **baseThreadIdx**

Base thread index (index of the first thread in the warp).

Indices of all other threads in the warp can be calculated by monotonically increasing the coordinates of the base thread index and wrapping around the block dimensions.

CuDim3 **blockIdx**

Block index.

uint64_t **gridId**

Grid ID.

uint32_t **validLanes**

Valid lanes (threads) mask.

uint32_t **warpAttributeFlags**

Bitmask of CUDBGWarpInfoAttribute_t enums for warps and their lanes.

3.74. CUDBGWarpResources

struct **CUDBGWarpResources**

Warp resources.

These resources can change at runtime between suspends.

Public Members

uint32_t **numRegisters**

Number of registers used by the warp.

uint32_t **sharedMemSize**

Shared memory size used by the warp.

3.75. CUDBGWarpState

struct **CUDBGWarpState**

Warp state information.

Public Members

uint32_t **activeLanes**

Lane mask of active threads in the warp.

CuDim3 **blockIdx**

Block index of the block containing the warp.

CuDim3 **clusterDim**

Cluster dimensions of the cluster containing the warp.

Can be different for different warps in the same grid.

CuDim3 **clusterExceptionTargetBlockIdx**

Cluster exception target block index of the the warp.

uint32_t **clusterExceptionTargetBlockIdxValid**

Whether the cluster exception target block index is valid.

CuDim3 **clusterIdx**

Cluster index of the cluster containing the warp.

uint64_t **errorPC**

Error PC of the warp (if any).

uint32_t **errorPCValid**

Whether the error PC is valid.

uint64_t **gridId**

Grid ID of the grid running in the warp.

uint32_t **inSyscallLanes**

Lane mask of threads in a syscall.

CUDBGLaneState **lane**[32]

State of the lanes (threads) in the warp.

uint32_t **validLanes**

Lane mask of valid threads in the warp.

3.76. CUDBGWarpState120

struct **CUDBGWarpState120**

Warp state for API version 12.0.

See also:

CUDBGWarpState

Note: DEPRECATED: Use *CUDBGWarpState* instead. For documentation of individual fields, see the documentation of the *CUDBGWarpState* struct instead.

Public Members

uint32_t **activeLanes**

CuDim3 **blockIdx**

CuDim3 **clusterIdx**

uint64_t **errorPC**

uint32_t **errorPCValid**

uint64_t **gridId**

CUDBGLaneState **lane**[32]

uint32_t **validLanes**

3.77. CUDBGWarpState127

struct **CUDBGWarpState127**

Warp state for API version 12.7.

See also:

CUDBGWarpState

Note: DEPRECATED: Use *CUDBGWarpState* instead. For documentation of individual fields, see the documentation of the *CUDBGWarpState* struct instead.

Public Members

uint32_t **activeLanes**

CuDim3 **blockIdx**

CuDim3 **clusterDim**

CuDim3 **clusterExceptionTargetBlockIdx**

uint32_t **clusterExceptionTargetBlockIdxValid**

CuDim3 **clusterIdx**

uint64_t **errorPC**

uint32_t **errorPCValid**

uint64_t **gridId**

CUDBGLaneState **lane**[32]

uint32_t **validLanes**

3.78. CUDBGWarpState60

struct **CUDBGWarpState60**

Warp state for API version 6.0.

See also:

CUDBGWarpState

Note: DEPRECATED: Use *CUDBGWarpState* instead. For documentation of individual fields, see the documentation of the *CUDBGWarpState* struct instead.

Public Members

uint32_t **activeLanes**

CuDim3 **blockIdx**

uint64_t **errorPC**

uint32_t **errorPCValid**

uint64_t **gridId**

CUDBGLaneState **lane**[32]

uint32_t **validLanes**

3.79. CuDim2

struct **CuDim2**

2-dimensional coordinates for threads, blocks, etc.

Note: DEPRECATED: Use *CuDim3* instead.

Public Members

uint32_t **x**

X coordinate.

uint32_t **y**

Y coordinate.

3.80. CuDim3

struct **CuDim3**

3-dimensional coordinates for threads, blocks, etc.

Public Members

uint32_t **x**

X coordinate.

uint32_t **y**

Y coordinate.

uint32_t **z**

Z coordinate.

CUDBGAPI_st

CUDA debugger API methods.

CUDBGAttributeValuePair

Grid attribute-value pair.

CUDBGCbuWarpState

Warp state in the CBU (Convergence Barrier Unit).

CUDBGCudaLogMessage

CUDA Log Message.

CUDBGCudaLogMessage129

CUDA Log Message for API version 12.9.

CUDBGCudaLogRule

CUDA Log filtering rule.

CUDBGDeviceInfo

Device-level information.

CUDBGDeviceInfoSizes

Sizes of the various structs returned by the batched device update APIs.

CUDBGEvent

Event information container.

CUDBGEvent30

Event information container for API version 3.0.

CUDBGEvent30::cases30_st::elfImageLoaded30_st

CUDBGEvent30::cases30_st::kernelFinished30_st

CUDBGEvent30::cases30_st::kernelReady30_st

CUDBGEvent32

Event information container for API version 3.2.

CUDBGEvent32::cases32_st::contextCreate32_st

CUDBGEvent32::cases32_st::contextDestroy32_st

CUDBGEvent32::cases32_st::contextPop32_st

CUDBGEvent32::cases32_st::contextPush32_st

CUDBGEvent32::cases32_st::elfImageLoaded32_st

CUDBGEvent32::cases32_st::kernelFinished32_st

CUDBGEvent32::cases32_st::kernelReady32_st

CUDBGEvent42
Event information container for API version 4.2.

CUDBGEvent42::cases42_st::contextCreate42_st

CUDBGEvent42::cases42_st::contextDestroy42_st

CUDBGEvent42::cases42_st::contextPop42_st

CUDBGEvent42::cases42_st::contextPush42_st

CUDBGEvent42::cases42_st::elfImageLoaded42_st

CUDBGEvent42::cases42_st::kernelFinished42_st

CUDBGEvent42::cases42_st::kernelReady42_st

CUDBGEvent50
Event information container for API version 5.0.

CUDBGEvent50::cases50_st::contextCreate50_st

CUDBGEvent50::cases50_st::contextDestroy50_st

CUDBGEvent50::cases50_st::contextPop50_st

CUDBGEvent50::cases50_st::contextPush50_st

CUDBGEvent50::cases50_st::elfImageLoaded50_st

CUDBGEvent50::cases50_st::internalError50_st

CUDBGEvent50::cases50_st::kernelFinished50_st

CUDBGEvent50::cases50_st::kernelReady50_st

CUDBGEvent55
Event information container for API version 5.5.

CUDBGEvent55::cases55_st::contextCreate55_st

CUDBGEvent55::cases55_st::contextDestroy55_st

CUDBGEvent55::cases55_st::contextPop55_st

CUDBGEvent55::cases55_st::contextPush55_st

CUDBGEvent55::cases55_st::elfImageLoaded55_st

CUDBGEvent55::cases55_st::internalError55_st

CUDBGEvent55::cases55_st::kernelFinished55_st

CUDBGEvent55::cases55_st::kernelReady55_st

CUDBGEvent::cases_st::allDevicesSuspended_st
Information about an event which forced all devices to be suspended.

CUDBGEvent::cases_st::contextCreate_st
Information about the context being created.

CUDBGEvent::cases_st::contextDestroy_st
Information about the context being destroyed.

CUDBGEvent::cases_st::contextPop_st
Information about the context being popped.

CUDBGEvent::cases_st::contextPush_st
Information about the context being pushed.

CUDBGEvent::cases_st::cudaLogsRulesetChanged_st
Information about the CUDA log ruleset being changed.

CUDBGEvent::cases_st::elfImageLoaded_st
Information about the loaded ELF image.

CUDBGEvent::cases_st::elfImageUnloaded_st
Information about the ELF image about to be unloaded.

CUDBGEvent::cases_st::functionsLoaded_st
Information about the functions being lazily loaded.

CUDBGEvent::cases_st::internalError_st
Information about internal erros.

CUDBGEvent::cases_st::kernelFinished_st
Information about the kernel that just terminated.

CUDBGEvent::cases_st::kernelReady_st
Information about the kernel ready to be launched.

CUDBGEvent::cases_st::singleStepComplete_st
Information about a single step operation that has completed.

CUDBGEventCallbackData
Callback data passed to callback set with setNotifyNewEventCallback function.

CUDBGEventCallbackData40
Callback data passed to callback set with setNotifyNewEventCallback40 function.

CUDBGEventCallbackData41
Callback data passed to callback set with setNotifyNewEventCallback41 function.

CUDBGGridInfo
Information about a CUDA grid.

CUDBGGridInfo120
Grid info.

CUDBGGridInfo55
Grid info.

CUDBGLaneInfo

CUDBGLaneState
Lane state (state of a single thread).

CUDBGLoadedFunctionInfo
Information about a lazily loaded function.

CUDBGMemoryInfo
Memory information.

CUDBGSMInfo
SM-level information.

CUDBGTcgen05TmemInfo
tcgen05 Tensor Memory information for a warp's CTA.

CUDBGWarpInfo
Warp-level information.

CUDBGWarpResources
Warp resources.

CUDBGWarpState
Warp state information.

CUDBGWarpState120
Warp state for API version 12.0.

CUDBGWarpState127
Warp state for API version 12.7.

CUDBGWarpState60
Warp state for API version 6.0.

CuDim2
2-dimensional coordinates for threads, blocks, etc.

CuDim3

3-dimensional coordinates for threads, blocks, etc.

Chapter 4. Data Fields

4.1. A

acknowledgeEvent30
CUDBGAPI_st

acknowledgeEvents42
CUDBGAPI_st

acknowledgeSyncEvents
CUDBGAPI_st

action
CUDBGCudaLogRule

activeLanes
CUDBGWarpInfo

CUDBGWarpState

CUDBGWarpState120

CUDBGWarpState127

CUDBGWarpState60

activeMask
CUDBGCbuWarpState

address
CUDBGLoadedFunctionInfo

allocationMask
CUDBGTcgen05TmemInfo

allocationPhase
CUDBGTcgen05TmemInfo

allocationStartMask
CUDBGTcgen05TmemInfo

attribute
CUDBGAttributeValuePair

4.2. B

barrierMasks
CUDBGCbuWarpState

baseThreadIdx
CUDBGWarpInfo

blockDim
CUDBGEvent42::cases42_st::kernelReady42_st
CUDBGEvent50::cases50_st::kernelReady50_st
CUDBGEvent55::cases55_st::kernelReady55_st
CUDBGEvent::cases_st::kernelReady_st
CUDBGGridInfo
CUDBGGridInfo120
CUDBGGridInfo55

blockIdx
CUDBGWarpInfo
CUDBGWarpState
CUDBGWarpState120
CUDBGWarpState127
CUDBGWarpState60

brokenDevicesMask
CUDBGEvent::cases_st::allDevicesSuspended_st
CUDBGEvent::cases_st::singleStepComplete_st

4.3. C

cases
CUDBGEvent
CUDBGEvent30
CUDBGEvent32
CUDBGEvent42
CUDBGEvent50
CUDBGEvent55

clearAttachState
CUDBGAPI_st

clusterDim
CUDBGGridInfo
CUDBGGridInfo120

CUDBGWarpState

CUDBGWarpState127

clusterExceptionTargetBlockIdx
CUDBGWarpState

CUDBGWarpState127

clusterExceptionTargetBlockIdxValid
CUDBGWarpState

CUDBGWarpState127

clusterIdx
CUDBGWarpState

CUDBGWarpState120

CUDBGWarpState127

collectiveMask
CUDBGCbuWarpState

consumeCudaLogs
CUDBGAPI_st

consumeCudaLogs129
CUDBGAPI_st

context
CUDBGEvent32::cases32_st::contextCreate32_st

CUDBGEvent32::cases32_st::contextDestroy32_st

CUDBGEvent32::cases32_st::contextPop32_st

CUDBGEvent32::cases32_st::contextPush32_st

CUDBGEvent32::cases32_st::elfImageLoaded32_st

CUDBGEvent32::cases32_st::kernelFinished32_st

CUDBGEvent32::cases32_st::kernelReady32_st

CUDBGEvent42::cases42_st::contextCreate42_st

CUDBGEvent42::cases42_st::contextDestroy42_st

CUDBGEvent42::cases42_st::contextPop42_st

CUDBGEvent42::cases42_st::contextPush42_st

CUDBGEvent42::cases42_st::elfImageLoaded42_st

CUDBGEvent42::cases42_st::kernelFinished42_st

CUDBGEvent42::cases42_st::kernelReady42_st

CUDBGEvent50::cases50_st::contextCreate50_st

CUDBGEvent50::cases50_st::contextDestroy50_st

CUDBGEvent50::cases50_st::contextPop50_st

CUDBGEvent50::cases50_st::contextPush50_st

CUDBGEvent50::cases50_st::elfImageLoaded50_st

CUDBGEvent50::cases50_st::kernelFinished50_st

CUDBGEvent50::cases50_st::kernelReady50_st

CUDBGEvent55::cases55_st::contextCreate55_st

CUDBGEvent55::cases55_st::contextDestroy55_st

CUDBGEvent55::cases55_st::contextPop55_st

CUDBGEvent55::cases55_st::contextPush55_st

CUDBGEvent55::cases55_st::elfImageLoaded55_st

CUDBGEvent55::cases55_st::kernelFinished55_st

CUDBGEvent55::cases55_st::kernelReady55_st

CUDBGEvent::cases_st::contextCreate_st

CUDBGEvent::cases_st::contextDestroy_st

CUDBGEvent::cases_st::contextPop_st

CUDBGEvent::cases_st::contextPush_st

CUDBGEvent::cases_st::elfImageLoaded_st

CUDBGEvent::cases_st::elfImageUnloaded_st

CUDBGEvent::cases_st::functionsLoaded_st

CUDBGEvent::cases_st::kernelFinished_st

CUDBGEvent::cases_st::kernelReady_st

CUDBGEvent::cases_st::singleStepComplete_st

CUDBGGridInfo

CUDBGGridInfo120

CUDBGGridInfo55

count
CUDBGEvent::cases_st::functionsLoaded_st

ctaGroup
CUDBGTcgen05TmemInfo

4.4. D

dev *CUDBGEvent30::cases30_st::kernelFinished30_st*

CUDBGEvent30::cases30_st::kernelReady30_st

CUDBGEvent32::cases32_st::contextCreate32_st

CUDBGEvent32::cases32_st::contextDestroy32_st

CUDBGEvent32::cases32_st::contextPop32_st

CUDBGEvent32::cases32_st::contextPush32_st

CUDBGEvent32::cases32_st::elfImageLoaded32_st

CUDBGEvent32::cases32_st::kernelFinished32_st
CUDBGEvent32::cases32_st::kernelReady32_st
CUDBGEvent42::cases42_st::contextCreate42_st
CUDBGEvent42::cases42_st::contextDestroy42_st
CUDBGEvent42::cases42_st::contextPop42_st
CUDBGEvent42::cases42_st::contextPush42_st
CUDBGEvent42::cases42_st::elfImageLoaded42_st
CUDBGEvent42::cases42_st::kernelFinished42_st
CUDBGEvent42::cases42_st::kernelReady42_st
CUDBGEvent50::cases50_st::contextCreate50_st
CUDBGEvent50::cases50_st::contextDestroy50_st
CUDBGEvent50::cases50_st::contextPop50_st
CUDBGEvent50::cases50_st::contextPush50_st
CUDBGEvent50::cases50_st::elfImageLoaded50_st
CUDBGEvent50::cases50_st::kernelFinished50_st
CUDBGEvent50::cases50_st::kernelReady50_st
CUDBGEvent55::cases55_st::contextCreate55_st
CUDBGEvent55::cases55_st::contextDestroy55_st
CUDBGEvent55::cases55_st::contextPop55_st
CUDBGEvent55::cases55_st::contextPush55_st
CUDBGEvent55::cases55_st::elfImageLoaded55_st
CUDBGEvent55::cases55_st::kernelFinished55_st
CUDBGEvent55::cases55_st::kernelReady55_st
CUDBGEvent::cases_st::contextCreate_st
CUDBGEvent::cases_st::contextDestroy_st
CUDBGEvent::cases_st::contextPop_st
CUDBGEvent::cases_st::contextPush_st
CUDBGEvent::cases_st::elfImageLoaded_st
CUDBGEvent::cases_st::elfImageUnloaded_st
CUDBGEvent::cases_st::functionsLoaded_st
CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::cases_st::kernelReady_st
CUDBGEvent::cases_st::singleStepComplete_st
CUDBGGridInfo
CUDBGGridInfo120
CUDBGGridInfo55

deviceAttributeFlags
CUDBGDeviceInfo

deviceInfoAttributeSizes
CUDBGDeviceInfoSizes

deviceInfoSize
CUDBGDeviceInfoSizes

disableBreakpoint
CUDBGAPI_st

disassemble
CUDBGAPI_st

4.5. E

elfImageHandle
CUDBGEvent::cases_st::functionsLoaded_st

CUDBGGridInfo

enableBreakpoint
CUDBGAPI_st

errorPC
CUDBGWarpState

CUDBGWarpState120

CUDBGWarpState127

CUDBGWarpState60

errorPCValid
CUDBGWarpState

CUDBGWarpState120

CUDBGWarpState127

CUDBGWarpState60

errorType
CUDBGEvent50::cases50_st::internalError50_st

CUDBGEvent55::cases55_st::internalError55_st

CUDBGEvent::cases_st::internalError_st

exception
CUDBGLaneState

executeInternalCommand
CUDBGAPI_st

exitedMask
CUDBGCbuWarpState

4.6. F

faultedDevicesMask
CUDBGEvent::cases_st::allDevicesSuspended_st
CUDBGEvent::cases_st::singleStepComplete_st

finalize
CUDBGAPI_st

finalWarpMask
CUDBGEvent::cases_st::singleStepComplete_st

function
CUDBGEvent32::cases32_st::kernelFinished32_st
CUDBGEvent32::cases32_st::kernelReady32_st
CUDBGEvent42::cases42_st::kernelFinished42_st
CUDBGEvent42::cases42_st::kernelReady42_st
CUDBGEvent50::cases50_st::kernelFinished50_st
CUDBGEvent50::cases50_st::kernelReady50_st
CUDBGEvent55::cases55_st::kernelFinished55_st
CUDBGEvent55::cases55_st::kernelReady55_st
CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::cases_st::kernelReady_st
CUDBGGridInfo
CUDBGGridInfo120
CUDBGGridInfo55

functionEntry
CUDBGEvent32::cases32_st::kernelFinished32_st
CUDBGEvent32::cases32_st::kernelReady32_st
CUDBGEvent42::cases42_st::kernelFinished42_st
CUDBGEvent42::cases42_st::kernelReady42_st
CUDBGEvent50::cases50_st::kernelFinished50_st
CUDBGEvent50::cases50_st::kernelReady50_st
CUDBGEvent55::cases55_st::kernelFinished55_st
CUDBGEvent55::cases55_st::kernelReady55_st
CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::cases_st::kernelReady_st
CUDBGGridInfo
CUDBGGridInfo120
CUDBGGridInfo55

4.7. G

generateCoredump
CUDBGAPI_st

getAdjustedCodeAddress
CUDBGAPI_st

getBindlessConstAddress
CUDBGAPI_st

getBlockDim
CUDBGAPI_st

getCbuWarpState
CUDBGAPI_st

getClusterDim
CUDBGAPI_st

getClusterDim120
CUDBGAPI_st

getClusterExceptionTargetBlock
CUDBGAPI_st

getConstBankAddress
CUDBGAPI_st

getConstBankAddress123
CUDBGAPI_st

getCudaExceptionString
CUDBGAPI_st

getDeviceInfo
CUDBGAPI_st

getDeviceInfoSizes
CUDBGAPI_st

getDeviceName
CUDBGAPI_st

getDevicePCIBusInfo
CUDBGAPI_st

getDeviceType
CUDBGAPI_st

getElfImage
CUDBGAPI_st

getElfImage32
CUDBGAPI_st

getElfImageByHandle
CUDBGAPI_st

getErrorStringEx
CUDBGAPI_st

getGridAttribute
CUDBGAPI_st

getGridAttributes
CUDBGAPI_st

getGridDim
CUDBGAPI_st

getGridDim32
CUDBGAPI_st

getGridInfo
CUDBGAPI_st

getGridInfo120
CUDBGAPI_st

getGridInfo55
CUDBGAPI_st

getGridStatus
CUDBGAPI_st

getGridStatus50
CUDBGAPI_st

getHardwareBarrierInfo
CUDBGAPI_st

getHostAddrFromDeviceAddr
CUDBGAPI_st

getLoadedFunctionInfo
CUDBGAPI_st

getLoadedFunctionInfo118
CUDBGAPI_st

getManagedMemoryRegionInfo
CUDBGAPI_st

getNextAsyncEvent50
CUDBGAPI_st

getNextAsyncEvent55
CUDBGAPI_st

getNextEvent
CUDBGAPI_st

getNextEvent30
CUDBGAPI_st

getNextEvent32
CUDBGAPI_st

getNextEvent42
CUDBGAPI_st

getNextSyncEvent50
CUDBGAPI_st

getNextSyncEvent55
CUDBGAPI_st

getNumDevices
CUDBGAPI_st

getNumLanes
CUDBGAPI_st

getNumPredicates
CUDBGAPI_st

getNumRegisters
CUDBGAPI_st

getNumSMs
CUDBGAPI_st

getNumUniformPredicates
CUDBGAPI_st

getNumUniformRegisters
CUDBGAPI_st

getNumWarps
CUDBGAPI_st

getPhysicalRegister30
CUDBGAPI_st

getPhysicalRegister40
CUDBGAPI_st

getSmType
CUDBGAPI_st

getSupportedDebuggerCapabilities
CUDBGAPI_st

getTID
CUDBGAPI_st

getWarpHitBreakpoint
CUDBGAPI_st

gridDim
CUDBGEvent42::cases42_st::kernelReady42_st

CUDBGEvent50::cases50_st::kernelReady50_st

CUDBGEvent55::cases55_st::kernelReady55_st

CUDBGEvent::cases_st::kernelReady_st

CUDBGGridInfo

CUDBGGridInfo120

CUDBGGridInfo55

gridId
CUDBGEvent30::cases30_st::kernelFinished30_st

CUDBGEvent30::cases30_st::kernelReady30_st

CUDBGEvent32::cases32_st::kernelFinished32_st

CUDBGEvent32::cases32_st::kernelReady32_st
CUDBGEvent42::cases42_st::kernelFinished42_st
CUDBGEvent42::cases42_st::kernelReady42_st
CUDBGEvent50::cases50_st::kernelFinished50_st
CUDBGEvent50::cases50_st::kernelReady50_st
CUDBGEvent55::cases55_st::kernelFinished55_st
CUDBGEvent55::cases55_st::kernelReady55_st
CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::cases_st::kernelReady_st
CUDBGWarpInfo
CUDBGWarpState
CUDBGWarpState120
CUDBGWarpState127
CUDBGWarpState60

gridId64
CUDBGEvent55::cases55_st::kernelFinished55_st
CUDBGEvent55::cases55_st::kernelReady55_st
CUDBGGridInfo
CUDBGGridInfo120
CUDBGGridInfo55

4.8. H

handle
CUDBGEvent::cases_st::elfImageLoaded_st
CUDBGEvent::cases_st::elfImageUnloaded_st

4.9. I

initialize
CUDBGAPI_st

initializeAttachStub
CUDBGAPI_st

insertBreakpoint
CUDBGAPI_st

inSyscallLanes
CUDBGWarpState

isBreakpointEnabled
CUDBGAPI_st

isDeviceCodeAddress
CUDBGAPI_st

isDeviceCodeAddress55
CUDBGAPI_st

4.10. K

kind
CUDBGEvent

CUDBGEvent30

CUDBGEvent32

CUDBGEvent42

CUDBGEvent50

CUDBGEvent55

4.11. L

lane *CUDBGWarpState*

CUDBGWarpState120

CUDBGWarpState127

CUDBGWarpState60

laneInfoAttributeSizes
CUDBGDeviceInfoSizes

laneInfoSize
CUDBGDeviceInfoSizes

logLevel
CUDBGCudaLogMessage

CUDBGCudaLogMessage129

logLevelFilter
CUDBGCudaLogRule

lookupDeviceCodeSymbol
CUDBGAPI_st

4.12. M

matchedRuleIndex
CUDBGCudaLogMessage

matchedRulesetIndex
CUDBGCudaLogMessage

memcheckReadErrorAddress
CUDBGAPI_st

message
CUDBGCudaLogMessage

CUDBGCudaLogMessage129

messageFilterRegex
CUDBGCudaLogRule

module
CUDBGEvent32::cases32_st::elfImageLoaded32_st

CUDBGEvent32::cases32_st::kernelFinished32_st

CUDBGEvent32::cases32_st::kernelReady32_st

CUDBGEvent42::cases42_st::elfImageLoaded42_st

CUDBGEvent42::cases42_st::kernelFinished42_st

CUDBGEvent42::cases42_st::kernelReady42_st

CUDBGEvent50::cases50_st::elfImageLoaded50_st

CUDBGEvent50::cases50_st::kernelFinished50_st

CUDBGEvent50::cases50_st::kernelReady50_st

CUDBGEvent55::cases55_st::elfImageLoaded55_st

CUDBGEvent55::cases55_st::kernelFinished55_st

CUDBGEvent55::cases55_st::kernelReady55_st

CUDBGEvent::cases_st::elfImageLoaded_st

CUDBGEvent::cases_st::elfImageUnloaded_st

CUDBGEvent::cases_st::kernelFinished_st

CUDBGEvent::cases_st::kernelReady_st

CUDBGGridInfo120

CUDBGGridInfo55

4.13. N

nonRelocatedElfImage
CUDBGEvent30::cases30_st::elfImageLoaded30_st
CUDBGEvent32::cases32_st::elfImageLoaded32_st
CUDBGEvent42::cases42_st::elfImageLoaded42_st
CUDBGEvent50::cases50_st::elfImageLoaded50_st
CUDBGEvent55::cases55_st::elfImageLoaded55_st

numRegisters
CUDBGWarpResources

4.14. O

origin
CUDBGEvent55::cases55_st::kernelReady55_st
CUDBGEvent::cases_st::kernelReady_st
CUDBGGridInfo
CUDBGGridInfo120
CUDBGGridInfo55

originalWarpMask
CUDBGEvent::cases_st::singleStepComplete_st

osThreadId
CUDBGCudaLogMessage
CUDBGCudaLogMessage129

osThreadIdFilter
CUDBGCudaLogRule

4.15. P

preferredClusterDim
CUDBGGridInfo

properties
CUDBGEvent::cases_st::elfImageLoaded_st

4.16. R

readActiveLanes
CUDBGAPI_st

readAllVirtualReturnAddresses
CUDBGAPI_st

readBlockIdx
CUDBGAPI_st

readBlockIdx32
CUDBGAPI_st

readBrokenWarps
CUDBGAPI_st

readCallDepth
CUDBGAPI_st

readCallDepth32
CUDBGAPI_st

readCCRegister
CUDBGAPI_st

readClusterIdx
CUDBGAPI_st

readCodeMemory
CUDBGAPI_st

readConstMemory129
CUDBGAPI_st

readCPUCallStack
CUDBGAPI_st

readDeviceExceptionState
CUDBGAPI_st

readDeviceExceptionState80
CUDBGAPI_st

readErrorPC
CUDBGAPI_st

readGenericMemory
CUDBGAPI_st

readGlobalMemory
CUDBGAPI_st

readGlobalMemory31
CUDBGAPI_st

readGlobalMemory55
CUDBGAPI_st

readGridId
CUDBGAPI_st

readGridId50
CUDBGAPI_st

readLaneException
CUDBGAPI_st

readLaneStatus
CUDBGAPI_st

readLocalMemory
CUDBGAPI_st

readParamMemory
CUDBGAPI_st

readPC
CUDBGAPI_st

readPinnedMemory
CUDBGAPI_st

readPredicates
CUDBGAPI_st

readRegister
CUDBGAPI_st

readRegisterRange
CUDBGAPI_st

readRegisterRange60
CUDBGAPI_st

readReturnAddress
CUDBGAPI_st

readReturnAddress32
CUDBGAPI_st

readRpcRegisters
CUDBGAPI_st

readSharedMemory
CUDBGAPI_st

readSmException
CUDBGAPI_st

readSyscallCallDepth
CUDBGAPI_st

readTcgen05TmemInfo
CUDBGAPI_st

readTcgen05TmemMemory
CUDBGAPI_st

readTextureMemory
CUDBGAPI_st

readTextureMemoryBindless
CUDBGAPI_st

readThreadIdx
CUDBGAPI_st

readUniformPredicates
CUDBGAPI_st

readUniformRegisterRange
CUDBGAPI_st

readValidLanes
CUDBGAPI_st

readValidWarps
CUDBGAPI_st

readVirtualPC
CUDBGAPI_st

readVirtualReturnAddress
CUDBGAPI_st

readVirtualReturnAddress32
CUDBGAPI_st

readWarpResources
CUDBGAPI_st

readWarpState
CUDBGAPI_st

readWarpState120
CUDBGAPI_st

readWarpState127
CUDBGAPI_st

readWarpState60
CUDBGAPI_st

relocatedElfImage
CUDBGEvent30::cases30_st::elfImageLoaded30_st

CUDBGEvent32::cases32_st::elfImageLoaded32_st

CUDBGEvent42::cases42_st::elfImageLoaded42_st

CUDBGEvent50::cases50_st::elfImageLoaded50_st

CUDBGEvent55::cases55_st::elfImageLoaded55_st

removeBreakpoint
CUDBGAPI_st

requestCleanupOnDetach
CUDBGAPI_st

requestCleanupOnDetach55
CUDBGAPI_st

requiredBufferSize
CUDBGDeviceInfoSizes

reserved0
CUDBGEvent55::cases55_st::kernelReady55_st

CUDBGEvent::cases_st::kernelReady_st

CUDBGGridInfo

CUDBGGridInfo120

CUDBGGridInfo55

responseType
CUDBGDeviceInfo

resumeAllDevices
CUDBGAPI_st

resumeDevice
CUDBGAPI_st

resumeWarpsUntilPC
CUDBGAPI_st

resumeWarpsUntilPC60
CUDBGAPI_st

rulesetIndex
CUDBGEvent::cases_st::cudaLogsRulesetChanged_st

4.17. S

sectionIndex
CUDBGLoadedFunctionInfo

setBreakpoint
CUDBGAPI_st

setBreakpoint31
CUDBGAPI_st

setCudaLogRules
CUDBGAPI_st

setKernelLaunchNotificationMode
CUDBGAPI_st

setNotifyNewEventCallback
CUDBGAPI_st

setNotifyNewEventCallback31
CUDBGAPI_st

setNotifyNewEventCallback40
CUDBGAPI_st

setNotifyNewEventCallback41
CUDBGAPI_st

sharedMemSize
CUDBGWarpResources

singleStepWarp
CUDBGAPI_st

singleStepWarp40
CUDBGAPI_st

singleStepWarp41
CUDBGAPI_st

singleStepWarp65
CUDBGAPI_st

size *CUDBGEvent30::cases30_st::elfImageLoaded30_st*

CUDBGEvent32::cases32_st::elfImageLoaded32_st

CUDBGEvent42::cases42_st::elfImageLoaded42_st

CUDBGEvent50::cases50_st::elfImageLoaded50_st

CUDBGEvent55::cases55_st::elfImageLoaded55_st

CUDBGEvent::cases_st::elfImageLoaded_st

CUDBGEvent::cases_st::elfImageUnloaded_st

CUDBGMemoryInfo

size32
CUDBGEvent42::cases42_st::elfImageLoaded42_st

CUDBGEvent50::cases50_st::elfImageLoaded50_st

CUDBGEvent55::cases55_st::elfImageLoaded55_st

sm *CUDBGEvent::cases_st::singleStepComplete_st*

smAttributeFlags
CUDBGSMInfo

smInfoAttributeSizes
CUDBGDeviceInfoSizes

smInfoSize
CUDBGDeviceInfoSizes

startAddress
CUDBGMemoryInfo

suspendAllDevices
CUDBGAPI_st

suspendDevice
CUDBGAPI_st

4.18. T

threadIdx
CUDBGLaneState

threadState
CUDBGCbuWarpState

tid *CUDBGEvent30::cases30_st::kernelFinished30_st*
CUDBGEvent30::cases30_st::kernelReady30_st
CUDBGEvent32::cases32_st::contextCreate32_st
CUDBGEvent32::cases32_st::contextDestroy32_st
CUDBGEvent32::cases32_st::contextPop32_st
CUDBGEvent32::cases32_st::contextPush32_st
CUDBGEvent32::cases32_st::kernelFinished32_st
CUDBGEvent32::cases32_st::kernelReady32_st
CUDBGEvent42::cases42_st::contextCreate42_st
CUDBGEvent42::cases42_st::contextDestroy42_st
CUDBGEvent42::cases42_st::contextPop42_st
CUDBGEvent42::cases42_st::contextPush42_st
CUDBGEvent42::cases42_st::kernelFinished42_st
CUDBGEvent42::cases42_st::kernelReady42_st
CUDBGEvent50::cases50_st::contextCreate50_st
CUDBGEvent50::cases50_st::contextDestroy50_st
CUDBGEvent50::cases50_st::contextPop50_st
CUDBGEvent50::cases50_st::contextPush50_st
CUDBGEvent50::cases50_st::kernelFinished50_st
CUDBGEvent50::cases50_st::kernelReady50_st
CUDBGEvent55::cases55_st::contextCreate55_st
CUDBGEvent55::cases55_st::contextDestroy55_st
CUDBGEvent55::cases55_st::contextPop55_st
CUDBGEvent55::cases55_st::contextPush55_st
CUDBGEvent55::cases55_st::kernelFinished55_st
CUDBGEvent55::cases55_st::kernelReady55_st
CUDBGEvent::cases_st::contextCreate_st
CUDBGEvent::cases_st::contextDestroy_st
CUDBGEvent::cases_st::contextPop_st
CUDBGEvent::cases_st::contextPush_st
CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::cases_st::kernelReady_st
CUDBGEventCallbackData
CUDBGEventCallbackData40
CUDBGEventCallbackData41
CUDBGGridInfo

CUDBGGridInfo120

CUDBGGridInfo55

timeout
CUDBGEventCallbackData41

type
CUDBGEvent42::cases42_st::kernelReady42_st

CUDBGEvent50::cases50_st::kernelReady50_st

CUDBGEvent55::cases55_st::kernelReady55_st

CUDBGEvent::cases_st::kernelReady_st

CUDBGEvent::cases_st::singleStepComplete_st

CUDBGGridInfo

CUDBGGridInfo120

CUDBGGridInfo55

4.19. U

unixTimestampNs
CUDBGCudaLogMessage

CUDBGCudaLogMessage129

unsetBreakpoint
CUDBGAPI_st

unsetBreakpoint31
CUDBGAPI_st

userData
CUDBGEventCallbackData

4.20. V

validLanes
CUDBGWarpInfo

CUDBGWarpState

CUDBGWarpState120

CUDBGWarpState127

CUDBGWarpState60

value
CUDBGAttributeValuePair

virtualPC
CUDBGLaneInfo

CUDBGLaneState

4.21. W

warpAttributeFlags
CUDBGWarpInfo

warpBrokenMask
CUDBGSMInfo

warpInfoAttributeSizes
CUDBGDeviceInfoSizes

warpInfoSize
CUDBGDeviceInfoSizes

warpValidMask
CUDBGSMInfo

writeCCRegister
CUDBGAPI_st

writeGenericMemory
CUDBGAPI_st

writeGlobalMemory
CUDBGAPI_st

writeGlobalMemory31
CUDBGAPI_st

writeGlobalMemory55
CUDBGAPI_st

writeLocalMemory
CUDBGAPI_st

writeParamMemory
CUDBGAPI_st

writePinnedMemory
CUDBGAPI_st

writePredicates
CUDBGAPI_st

writeRegister
CUDBGAPI_st

writeRpcRegisters
CUDBGAPI_st

writeSharedMemory
CUDBGAPI_st

writeTcgen05TmemMemory
CUDBGAPI_st

writeUniformPredicates
CUDBGAPI_st

writeUniformRegister
CUDBGAPI_st

4.22. X

x *CuDim2*
CuDim3

4.23. Y

y *CuDim2*
CuDim3

4.24. Z

z *CuDim3*

Chapter 5. Release Notes

5.1. CUDA 13.4 Release

5.1.1. Deprecated APIs

- CUDBG_EVENT_KERNEL_READY and CUDBG_EVENT_KERNEL_FINISHED are deprecated because they are not reliable and can't be sent for all kernel executions.
- kernelReady_st, kernelFinished_st and their legacy versions are deprecated.

5.1.2. Dropped APIs

- parentGridId is no longer exposed. To keep ABI compatibility, it has been replaced with reserved0 in kernelReady_st, CUDBGGridInfo and their legacy versions.

5.2. CUDA 13.3 Release

5.2.1. New APIs

- CUDBG_EXCEPTION_TMA_SYSCALL: Report TMA syscall errors.
- CUDBG_COREDUMP_FAULTED_CONTEXTS_ONLY: Include only faulted contexts in CUDA core dumps.

5.3. CUDA 13.2 Release

5.3.1. Major changes

- Added handle-based breakpoint management.
- Added coordinated multi-device suspension and resumption.
- Added non-blocking single-step support.

5.3.2. New APIs

- insertBreakpoint(), removeBreakpoint(), enableBreakpoint(), disableBreakpoint(), isBreakpointEnabled(), getWarpHitBreakpoint(): Manage and query breakpoints using CUDBGBreakpointHandle.
- CUDBG_DEBUGGER_CAPABILITY_BREAK_ON_LAUNCH: Enable break on launch.
- resumeWarpsUntilPC(): Add a flags parameter, including non-blocking operation with CUDBG_SINGLE_STEP_FLAGS_NON_BLOCKING.
- CUDBG_EVENT_SINGLE_STEP_COMPLETE: Report completion of non-blocking stepping.
- suspendAllDevices(), resumeAllDevices(): Suspend or resume all CUDA devices.
- readRegisterRange(): Add an optional numRegistersRead output parameter that reports the number of registers read.

5.3.3. Deprecated APIs

- resumeWarpsUntilPC60(); use resumeWarpsUntilPC() instead.
- readRegisterRange60(); use readRegisterRange() instead.

5.4. CUDA 13.1 Release

5.4.1. Major changes

- Removed memcheck-specific exceptions and CUDBG_ERROR_MEMCHECK_NOT_ENABLED, reserving their values for ABI compatibility.
- Added hardware barrier, warp and cluster exception reporting, gzip core dumps, and file-descriptor-based debugger attach.

5.4.2. New APIs

- getHardwareBarrierInfo(): Get hardware barrier information for a thread.
- CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD, CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED: Initiate debugger attach through a file descriptor and report completion.
- CUDBG_DEBUGGER_CAPABILITY_FLUSH_PRINTF_ON_SUSPEND: Flush CUDA printf output when a device is suspended.
- CUDBG_COREDUMP_GZIP_COMPRESS: Compress CUDA core dumps with gzip.
- CudbgWarpTableEntry::barrierScope, CudbgWarpTableEntry::additionalBarrierInfo (cudacore-dump.h): Record hardware barrier information in core dumps.

5.4.3. Deprecated APIs

- readCCRegister()
- writeCCRegister()

5.4.4. Dropped APIs

- cudbgMain(); use CUDBG_PRE_INIT instead.
- CUDBG_DETACH_SUSPENDED_DEVICES_MASK, CUDBG_ENABLE_INTEGRATED_MEMCHECK, and CUDBG_ENABLE_PREEMPTION_DEBUGGING

5.5. CUDA 13.0 Release

5.5.1. Major changes

- Removed support for Maxwell, Pascal and Volta architectures.
- Removed support for CUDA_ENABLE_LIGHTWEIGHT_COREDUMP.
- Reduced CPU call stack collection overhead and disabled it by default.
- Bugfixes and performance improvements.

5.5.2. New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getCudaExceptionString(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, char *buf, uint32_t bufSz, uint32_t *msgSz)
Get error string for CUDA Exceptions.

setNotifyNewEventCallback(CUDBGNotifyNewEventCallback callback, void* userData)
Provides the API with the function to call to notify the debugger of a new application or device event.

CUDBG_DEBUGGER_CAPABILITY_COLLECT_CPU_CALL_STACK_FOR_KERNEL_LAUNCHES
New capability to collect CPU call stack for kernel launches. The readCPUCallStack can only be used when this capability is enabled.

5.5.3. Deprecated APIs

- readConstMemory(uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

5.6. CUDA 12.9 Release

5.6.1. Major changes

- Support late attach on WSL.
- Bugfixes and performance improvements.

5.6.2. New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getCbuWarpState(uint32_t dev, uint32_t sm, uint64_t warpMask, CUDBGCbuWarpState* warpStates, uint32_t numWarpStates)
Gets CBU state of a given warp.

consumeCudaLogs(CUDBGCudaLogMessage* logMessages, uint32_t numMessages, uint32_t* numConsumed)
Get CUDA error log entries. This consumes the log entries, so they will not be available in subsequent calls.

readCPUCallStack(uint32_t dev, uint64_t gridId64, uint64_t *addrs, uint32_t numAddrs, uint32_t* totalNumAddrs)
Read CPU call stack captured at the time of kernel launch.

5.6.3. Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- readWarpState(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState *state)

5.7. CUDA 12.8 Release

5.7.1. Major changes

- Support late attach on Jetson targets.
- Bugfixes and performance improvements.

5.7.2. New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

readWarpResources(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpResources *resources)
Get the resources assigned to a given warp.

5.8. CUDA 12.7 Release

5.8.1. Major changes

- Bugfixes and performance improvements.

5.8.2. New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getClusterExceptionTargetBlock(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockIdx, bool *blockIdxValid)
Retrieves the target block index and validity status for a given device, streaming multiprocessor, and warp for cluster exceptions.

5.8.3. Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- getGridInfo(uint32_t dev, uint64_t gridId64, CUDBGGridInfo *gridInfo)
- getClusterDim(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *clusterDim)
- readWarpState(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState *state)

5.9. CUDA 12.6 Release

5.9.1. Major changes

- Reduced overhead for batch breakpoint updates.
- Moved constbank memory dump control to a separate flag.
- Bugfixes and performance improvements.

5.9.2. New APIs

CUDBG_COREDUMP_SKIP_CONSTBANK_MEMORY flag for coredump generation (generateCoredump).
In the previous API versions the CUDBG_COREDUMP_SKIP_GLOBAL_MEMORY controlled both the global and the constant memory. Since CUDA 12.6 the constbank memory is controlled by a separate flag (the constbank memory is usually smaller than global memory, so it is feasible to dump it by default even if the global memory is skipped).

5.10. CUDA 12.5 Release

5.10.1. Major changes

- Significant performance improvements for memory accessing operations and stepping.
- Bugfixes and stability improvements.

5.10.2. New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

readAllVirtualReturnAddresses(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *addrs, uint32_t numAddrs, uint32_t* callDepth, uint32_t* syscallCallDepth)
Reads all the virtual return addresses.

getSupportedDebuggerCapabilities(CUDBGCapabilityFlags* capabilities)
Returns debugger capabilities that are supported by this version of the API.

readSmException(uint32_t dev, uint32_t sm, CUDBGException_t *exception, uint64_t *errorPC, bool *errorPCValid)
Get the SM exception status if it exists.

5.10.3. Deprecated APIs

CUDBG_COREDUMP_SKIP_ABORT
The generateCoredump API no longer accepts the CUDBG_COREDUMP_SKIP_ABORT flag. Note that this flag was ignored in the previous versions of the API.

5.11. CUDA 12.4 Release

5.11.1. Major changes

- Made CUDA ELF file handling more robust and performant.
- Bugfixes and performance improvements.

5.11.2. New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getDeviceInfoSizes(uint32_t dev, CUDBGDeviceInfoSizes* sizes)
Returns sizes for device info structs and defined attributes.

getDeviceInfo(uint32_t dev, CUDBGDeviceInfoQueryType_t type, void *buffer, uint32_t length, uint32_t *dataLength)
Returns full or changed device info.

5.11.3. Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- getConstBankAddress(uint32_t dev, uint64_t gridId64, uint32_t bank, uint64_t* address, uint32_t* size)
- singleStepWarp(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t laneHint, uint32_t nsteps, uint32_t flags, uint64_t *warpMask)

5.12. CUDA 12.3 Release

5.12.1. Major changes

- Support generating coredumps after the debugger has attached.
- Bugfixes and performance improvements.

5.12.2. New APIs

getConstBankAddress(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t bank, uint32_t offset, uint64_t* address)
Returns sizes for device info structs and defined attributes.

generateCoredump(const char* filename, CUDBGCoredumpGenerationFlags flags)
Generates a coredump for the current GPU state.

5.12.3. Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- getLoadedFunctionInfo(uint32_t devId, uint64_t handle, CUDBGLoadedFunctionInfo *info, uint32_t startIndex, uint32_t numEntries)

5.12.4. Deprecated APIs

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

5.13. CUDA 12.2 Release

5.13.1. Major changes

- Switch to the new debugger back-end (Unified Debugger) on WSL.
- Switch to the new debugger back-end (Unified Debugger) on Jetson.
- Bugfixes and performance improvements.

5.13.2. New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getErrorStringEx(char *buf, uint32_t bufSz, uint32_t *msgSz)
Fills a user-provided buffer with an error message encoded as a null-terminated ASCII string. The error message is specific to the last failed API call and is invalidated after every API call.

5.14. CUDA 12.1 Release

5.14.1. Major changes

- Improved support for single stepping.
- Bugfixes and performance improvements.

5.15. CUDA 12.0 Release

5.15.1. Major changes

- Debugging support for application using CUDA Dynamic Parallelism V2.
- Improved support for latest GPU architectures.
- Bugfixes and performance improvements.

5.15.2. New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getClusterDim(uint32_t dev, uint64_t gridId64, CuDim3 *clusterDim)
Get the number of blocks in the given cluster.

readClusterIdx(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *clusterIdx)
Get the number of blocks in the given cluster.

5.15.3. Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- getGridInfo(uint32_t dev, uint64_t gridId64, CUDBGGridInfo *gridInfo)
- readWarpState(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState *state)

5.16. CUDA 11.8 Release

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

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

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

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

5.17. CUDA 7.0 Release

Stability improvements. No API additions or changes.

5.18. CUDA 6.5 Release

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

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

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

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

Chapter 6. Notices

6.1. Notice

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

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

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

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

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or

services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND 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. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

6.2. OpenCL

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

6.3. Trademarks

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

Copyright

©2007-2026, NVIDIA Corporation & affiliates. All rights reserved