

# AMD GPU Debug API Specification

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Overview . . . . .	1
1.2	Assumptions . . . . .	1
1.3	Requirements . . . . .	1
<b>2</b>	<b>Data Structure Index</b>	<b>3</b>
2.1	Data Structures . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Data Structure Documentation</b>	<b>7</b>
4.1	HwDbgDataBreakpointInfo Struct Reference . . . . .	7
4.1.1	Detailed Description . . . . .	7
4.1.2	Field Documentation . . . . .	7
4.1.2.1	dataBreakpointMode . . . . .	7
4.1.2.2	dataSize . . . . .	7
4.1.2.3	pAddress . . . . .	8
4.2	HwDbgDim3 Struct Reference . . . . .	8
4.2.1	Detailed Description . . . . .	8
4.2.2	Field Documentation . . . . .	8
4.2.2.1	x . . . . .	8
4.2.2.2	y . . . . .	8
4.2.2.3	z . . . . .	8
4.3	HwDbgState Struct Reference . . . . .	9

4.3.1	Detailed Description	9
4.3.2	Field Documentation	9
4.3.2.1	behaviorFlags	9
4.3.2.2	packetId	9
4.3.2.3	pDevice	9
4.3.2.4	pPacket	9
4.4	HwDbgWavefrontInfo Struct Reference	10
4.4.1	Detailed Description	10
4.4.2	Field Documentation	10
4.4.2.1	breakpointType	10
4.4.2.2	codeAddress	11
4.4.2.3	dataBreakpointHandle	11
4.4.2.4	executionMask	11
4.4.2.5	pOtherData	11
4.4.2.6	wavefrontAddress	11
4.4.2.7	workGroupId	11
4.4.2.8	workItemId	11
<b>5</b>	<b>File Documentation</b>	<b>13</b>
5.1	AMDGPUDebug.h File Reference	13
5.1.1	Detailed Description	15
5.1.2	Macro Definition Documentation	15
5.1.2.1	AMDGPUDEBUG_VERSION_BUILD	15
5.1.2.2	AMDGPUDEBUG_VERSION_MAJOR	16
5.1.2.3	AMDGPUDEBUG_VERSION_MINOR	16
5.1.2.4	HWDBG_API_CALL	16
5.1.2.5	HWDBG_API_ENTRY	16
5.1.2.6	HWDBG_WAVEFRONT_SIZE	16
5.1.3	Typedef Documentation	16
5.1.3.1	HwDbgCodeAddress	16
5.1.3.2	HwDbgCodeBreakpointHandle	16

5.1.3.3	HwDbgContextHandle . . . . .	17
5.1.3.4	HwDbgDataBreakpointHandle . . . . .	17
5.1.3.5	HwDbgLoggingCallback . . . . .	17
5.1.3.6	HwDbgWavefrontAddress . . . . .	17
5.1.4	Enumeration Type Documentation . . . . .	17
5.1.4.1	HwDbgAPIType . . . . .	17
5.1.4.2	HwDbgBehaviorType . . . . .	18
5.1.4.3	HwDbgBreakpointType . . . . .	18
5.1.4.4	HwDbgCommand . . . . .	18
5.1.4.5	HwDbgDataBreakpointMode . . . . .	18
5.1.4.6	HwDbgEventType . . . . .	19
5.1.4.7	HwDbgLogType . . . . .	19
5.1.4.8	HwDbgStatus . . . . .	19
5.1.5	Function Documentation . . . . .	20
5.1.5.1	HwDbgBeginDebugContext(const HwDbgState state, HwDbgContextHandle *pDebugContextOut) . . . . .	20
5.1.5.2	HwDbgBreakAll(const HwDbgContextHandle hDebugContext) . . . . .	20
5.1.5.3	HwDbgContinueEvent(HwDbgContextHandle hDebugContext, const HwDbg↵ Command command) . . . . .	21
5.1.5.4	HwDbgCreateCodeBreakpoint(HwDbgContextHandle hDebugContext, const HwDbgCodeAddress codeAddress, HwDbgCodeBreakpointHandle *p↵ BreakpointOut) . . . . .	21
5.1.5.5	HwDbgCreateDataBreakpoint(HwDbgContextHandle hDebugContext, const HwDbgDataBreakpointInfo breakpointInfo, HwDbgDataBreakpointHandle *p↵ DataBreakpointOut) . . . . .	22
5.1.5.6	HwDbgDeleteAllCodeBreakpoints(HwDbgContextHandle hDebugContext) . . . . .	23
5.1.5.7	HwDbgDeleteAllDataBreakpoints(HwDbgContextHandle hDebugContext) . . . . .	24
5.1.5.8	HwDbgDeleteCodeBreakpoint(HwDbgContextHandle hDebugContext, HwDbg↵ CodeBreakpointHandle hBreakpoint) . . . . .	24
5.1.5.9	HwDbgDeleteDataBreakpoint(HwDbgContextHandle hDebugContext, HwDbg↵ DataBreakpointHandle hDataBreakpoint) . . . . .	25
5.1.5.10	HwDbgEndDebugContext(HwDbgContextHandle hDebugContext) . . . . .	26
5.1.5.11	HwDbgGetActiveWavefronts(const HwDbgContextHandle hDebugContext, const HwDbgWavefrontInfo **ppWavefrontInfoOut, uint32_t *pNumWavefrontsOut) . . . . .	26

5.1.5.12	HwDbgGetAPIType(HwDbgAPIType *pAPITypeOut) . . . . .	27
5.1.5.13	HwDbgGetAPIVersion(uint32_t *pVersionMajorOut, uint32_t *pVersionMinorOut, uint32_t *pVersionBuildOut) . . . . .	27
5.1.5.14	HwDbgGetCodeBreakpointAddress(const HwDbgContextHandle hDebugContext, const HwDbgCodeBreakpointHandle hBreakpoint, HwDbgCodeAddress *pCodeAddressOut) . . . . .	28
5.1.5.15	HwDbgGetDataBreakpointInfo(const HwDbgContextHandle hDebugContext, const HwDbgDataBreakpointHandle hDataBreakpoint, HwDbgDataBreakpointInfo *pDataBreakpointInfoOut) . . . . .	28
5.1.5.16	HwDbgGetDispatchedKernelName(const HwDbgContextHandle hDebugContext, const char **ppKernelNameOut) . . . . .	29
5.1.5.17	HwDbgGetKernelBinary(const HwDbgContextHandle hDebugContext, const void **ppBinaryOut, size_t *pBinarySizeOut) . . . . .	30
5.1.5.18	HwDbgInit(void *pApiTable) . . . . .	31
5.1.5.19	HwDbgKillAll(const HwDbgContextHandle hDebugContext) . . . . .	31
5.1.5.20	HwDbgReadMemory(const HwDbgContextHandle hDebugContext, const uint32_t memoryRegion, const HwDbgDim3 workGroupId, const HwDbgDim3 workItemId, const size_t offset, const size_t numBytesToRead, void *pMemOut, size_t *pNumBytesOut) . . . . .	32
5.1.5.21	HwDbgSetLoggingCallback(uint32_t types, HwDbgLoggingCallback pCallback, void *pUserData) . . . . .	32
5.1.5.22	HwDbgShutDown() . . . . .	33
5.1.5.23	HwDbgWaitForEvent(HwDbgContextHandle hDebugContext, const uint32_t timeout, HwDbgEventType *pEventTypeOut) . . . . .	33
<b>Index</b>		<b>35</b>

# Chapter 1

## Introduction

### 1.1 Overview

This document describes a set of interfaces which can be used by debugger or application developers to incorporate GPU kernel debugging functionality into their debugger or application running on AMD Graphics Core Next GPUs (or APUs).

The AMD GPU Kernel Debugging API has been designed to hide the multiple driver API specific implementations and the internal architecture of a particular GPU device. It has evolved starting from a minimal set of GPU debugging APIs that can be currently supported by AMD GPUs and software stacks. As more GPU debug features are implemented and validated, the API will evolve further. It is still a work-in-progress.

For HSA, this API together with the AMD HSA binary interface, AMD HSA debug info and AMD HSA API and dispatch interception mechanism form the AMD HSA GPU Debugging Architecture. Refer to the "AMD HSA GPU Debugging Architecture" document for more information.

### 1.2 Assumptions

The AMD GPU Kernel Debugging API is an "in-process" debug API. That is, the API must be called from the same process address space as the program being debugged and will have direct access to all process resources. No OS provided inter-process debug mechanisms are required, but it should be reasonably straightforward for tool developers to create a client/server remote debugging model through the introduction of a simple communication protocol.

To inject these kernel debugging API calls into the debugged application process address space, the API and kernel dispatch interception mechanism provided through `amd_hsa_tools_interfaces.h` can be used.

### 1.3 Requirements

For HSA:

1. AMD Kaveri and Carrizo APUs
2. HSA Runtime and HSAIL 1.0 Final





## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">HwDbgDataBreakpointInfo</a>	7
<a href="#">HwDbgDim3</a>	8
<a href="#">HwDbgState</a>	9
<a href="#">HwDbgWavefrontInfo</a>	10



## Chapter 3

# File Index

### 3.1 File List

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

[AMDGPUDebug.h](#)

The AMD GPU Kernel Debugging API to implement device kernel debugging on AMD Graphics  
Core Next (GCN) GPUs . . . . .

13



## Chapter 4

# Data Structure Documentation

### 4.1 HwDbgDataBreakpointInfo Struct Reference

```
#include <AMDGPUDebug.h>
```

#### Data Fields

- [HwDbgDataBreakpointMode](#) dataBreakpointMode
- [uint64\\_t](#) dataSize
- [void \\*](#) pAddress

#### 4.1.1 Detailed Description

A structure to hold all the info required to create a single data breakpoint.

##### Warning

This is not yet supported

Definition at line 229 of file AMDGPUDebug.h.

#### 4.1.2 Field Documentation

##### 4.1.2.1 HwDbgDataBreakpointMode HwDbgDataBreakpointInfo::dataBreakpointMode

the relevant mode for the data breakpoint

Definition at line 232 of file AMDGPUDebug.h.

##### 4.1.2.2 uint64\_t HwDbgDataBreakpointInfo::dataSize

the size of data in bytes being watched

Definition at line 235 of file AMDGPUDebug.h.

#### 4.1.2.3 void\* HwDbgDataBreakpointInfo::pAddress

the memory address to be watched

Definition at line 238 of file AMDGPUDebug.h.

## 4.2 HwDbgDim3 Struct Reference

```
#include <AMDGPUDebug.h>
```

### Data Fields

- uint32\_t [x](#)
- uint32\_t [y](#)
- uint32\_t [z](#)

#### 4.2.1 Detailed Description

A three dimensional type, used by work-group and work-item ids.

Definition at line 220 of file AMDGPUDebug.h.

#### 4.2.2 Field Documentation

##### 4.2.2.1 uint32\_t HwDbgDim3::x

x dimension

Definition at line 222 of file AMDGPUDebug.h.

##### 4.2.2.2 uint32\_t HwDbgDim3::y

y dimension

Definition at line 223 of file AMDGPUDebug.h.

##### 4.2.2.3 uint32\_t HwDbgDim3::z

z dimension

Definition at line 224 of file AMDGPUDebug.h.

## 4.3 HwDbgState Struct Reference

```
#include <AMDGPUDebug.h>
```

### Data Fields

- void \* [pDevice](#)
- void \* [pPacket](#)
- uint64\_t [packetId](#)
- uint32\_t [behaviorFlags](#)

#### 4.3.1 Detailed Description

A structure to hold the device state as an input to the HwDbgBeginDebugContext

Definition at line 286 of file AMDGPUDebug.h.

#### 4.3.2 Field Documentation

##### 4.3.2.1 uint32\_t HwDbgState::behaviorFlags

flags that the control the behavior of the debug context

Definition at line 299 of file AMDGPUDebug.h.

##### 4.3.2.2 uint64\_t HwDbgState::packetId

set to packet\_id from the pre-dispatch callback function

#### Warning

This is not yet supported

Definition at line 296 of file AMDGPUDebug.h.

##### 4.3.2.3 void\* HwDbgState::pDevice

set to hsa\_agent\_t.handle from the pre-dispatch callback function

Definition at line 289 of file AMDGPUDebug.h.

##### 4.3.2.4 void\* HwDbgState::pPacket

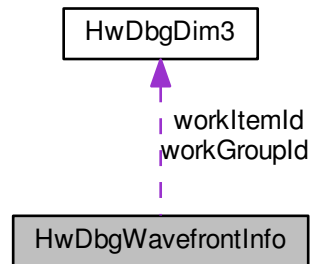
set to hsa\_kernel\_dispatch\_packet\_t\* from the pre-dispatch callback function

Definition at line 292 of file AMDGPUDebug.h.

## 4.4 HwDbgWavefrontInfo Struct Reference

```
#include <AMDGPUDebug.h>
```

Collaboration diagram for HwDbgWavefrontInfo:



### Data Fields

- [HwDbgDim3](#) **workGroupId**
- [HwDbgDim3](#) **workItemId** [HWDBG\_WAVEFRONT\_SIZE]
- [uint64\\_t](#) **executionMask**
- [HwDbgWavefrontAddress](#) **wavefrontAddress**
- [HwDbgCodeAddress](#) **codeAddress**
- [HwDbgDataBreakpointHandle](#) **dataBreakpointHandle**
- [HwDbgBreakpointType](#) **breakpointType**
- [void \\*](#) **pOtherData**

### 4.4.1 Detailed Description

A structure to hold the active wave info returned by HwDbgGetActiveWavefronts API

Definition at line 242 of file AMDGPUDebug.h.

### 4.4.2 Field Documentation

#### 4.4.2.1 HwDbgBreakpointType HwDbgWavefrontInfo::breakpointType

the type of breakpoint that was signaled

#### Warning

This is not yet supported

Definition at line 265 of file AMDGPUDebug.h.



#### 4.4.2.2 HwDbgCodeAddress HwDbgWavefrontInfo::codeAddress

the byte offset in the ISA binary for the wavefront

Definition at line 257 of file AMDGPUDebug.h.

#### 4.4.2.3 HwDbgDataBreakpointHandle HwDbgWavefrontInfo::dataBreakpointHandle

the data breakpoint handle

##### Warning

This is not yet supported

Definition at line 261 of file AMDGPUDebug.h.

#### 4.4.2.4 uint64\_t HwDbgWavefrontInfo::executionMask

the execution mask of the work-items

Definition at line 251 of file AMDGPUDebug.h.

#### 4.4.2.5 void\* HwDbgWavefrontInfo::pOtherData

additional data that can be returned

Definition at line 268 of file AMDGPUDebug.h.

#### 4.4.2.6 HwDbgWavefrontAddress HwDbgWavefrontInfo::wavefrontAddress

the hardware wavefront slot address (not unique for a dispatch)

Definition at line 254 of file AMDGPUDebug.h.

#### 4.4.2.7 HwDbgDim3 HwDbgWavefrontInfo::workGroupId

the work-group id

Definition at line 245 of file AMDGPUDebug.h.

#### 4.4.2.8 HwDbgDim3 HwDbgWavefrontInfo::workItemId[HWDBG\_WAVEFRONT\_SIZE]

the work-item id (local id within a work-group)

Definition at line 248 of file AMDGPUDebug.h.



## Chapter 5

# File Documentation

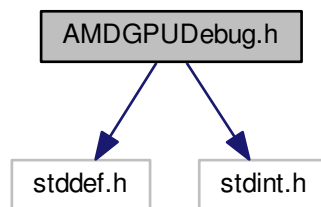
### 5.1 AMDGPUDebug.h File Reference

The AMD GPU Kernel Debugging API to implement device kernel debugging on AMD Graphics Core Next (GCN) GPUs.

```
#include <stddef.h>
```

```
#include <stdint.h>
```

Include dependency graph for AMDGPUDebug.h:



#### Data Structures

- struct [HwDbgDim3](#)
- struct [HwDbgDataBreakpointInfo](#)
- struct [HwDbgWavefrontInfo](#)
- struct [HwDbgState](#)

#### Macros

- #define [HWDBG\\_API\\_ENTRY](#)
- #define [HWDBG\\_API\\_CALL](#)
- #define [AMDGPUDEBUG\\_VERSION\\_MAJOR](#) 1
- #define [AMDGPUDEBUG\\_VERSION\\_MINOR](#) 3
- #define [AMDGPUDEBUG\\_VERSION\\_BUILD](#) 5
- #define [HWDBG\\_WAVEFRONT\\_SIZE](#) 64

## Typedefs

- typedef uint64\_t [HwDbgCodeAddress](#)
- typedef uint32\_t [HwDbgWavefrontAddress](#)
- typedef void \* [HwDbgContextHandle](#)
- typedef void \* [HwDbgCodeBreakpointHandle](#)
- typedef void \* [HwDbgDataBreakpointHandle](#)
- typedef void(\* [HwDbgLoggingCallback](#)) (void \*pUserData, const [HwDbgLogType](#) type, const char \*const pMessage)

## Enumerations

- enum [HwDbgStatus](#) {  
[HWDBG\\_STATUS\\_SUCCESS](#) = 0x0, [HWDBG\\_STATUS\\_ERROR](#) = 0x01, [HWDBG\\_STATUS\\_DEVICE\\_](#)↵  
[ERROR](#) = 0x02, [HWDBG\\_STATUS\\_DRIVER\\_ERROR](#) = 0x03,  
[HWDBG\\_STATUS\\_DUPLICATE\\_BREAKPOINT](#) = 0x04, [HWDBG\\_STATUS\\_INVALID\\_ADDRESS\\_ALIG](#)↵  
[NMENT](#) = 0x05, [HWDBG\\_STATUS\\_INVALID\\_HANDLE](#) = 0x06, [HWDBG\\_STATUS\\_INVALID\\_PARAMET](#)↵  
[ER](#) = 0x07,  
[HWDBG\\_STATUS\\_NULL\\_POINTER](#) = 0x08, [HWDBG\\_STATUS\\_OUT\\_OF\\_RANGE\\_ADDRESS](#) = 0x09,  
[HWDBG\\_STATUS\\_OUT\\_OF\\_MEMORY](#) = 0x0A, [HWDBG\\_STATUS\\_OUT\\_OF\\_RESOURCES](#) = 0x0B,  
[HWDBG\\_STATUS\\_REGISTRATION\\_ERROR](#) = 0x0C, [HWDBG\\_STATUS\\_UNDEFINED](#) = 0x0D, [HWDB](#)↵  
[G\\_STATUS\\_UNSUPPORTED](#) = 0x0E, [HWDBG\\_STATUS\\_NOT\\_INITIALIZED](#) = 0x0F,  
[HWDBG\\_STATUS\\_INVALID\\_BEHAVIOR\\_STATE](#) = 0x10 }
- enum [HwDbgCommand](#) { [HWDBG\\_COMMAND\\_CONTINUE](#) = 0x0 }
- enum [HwDbgAPIType](#) { [HWDBG\\_API\\_HSA](#) = 0x0 }
- enum [HwDbgBreakpointType](#) { [HWDBG\\_BREAKPOINT\\_TYPE\\_NONE](#) = 0x0, [HWDBG\\_BREAKPOINT\\_T](#)↵  
[YPE\\_CODE](#) = 0x1, [HWDBG\\_BREAKPOINT\\_TYPE\\_DATA](#) = 0x2 }
- enum [HwDbgEventType](#) { [HWDBG\\_EVENT\\_INVALID](#) = 0x0, [HWDBG\\_EVENT\\_TIMEOUT](#) = 0x1, [HWDBG](#)↵  
[\\_EVENT\\_POST\\_BREAKPOINT](#) = 0x2, [HWDBG\\_EVENT\\_END\\_DEBUGGING](#) = 0x3 }
- enum [HwDbgDataBreakpointMode](#) { [HWDBG\\_DATABREAKPOINT\\_MODE\\_READ](#) = 0x1, [HWDBG\\_DAT](#)↵  
[ABREAKPOINT\\_MODE\\_NONREAD](#) = 0x2, [HWDBG\\_DATABREAKPOINT\\_MODE\\_ATOMIC](#) = 0x4, [HWD](#)↵  
[BG\\_DATABREAKPOINT\\_MODE\\_ALL](#) = 0x7 }
- enum [HwDbgBehaviorType](#) { [HWDBG\\_BEHAVIOR\\_NONE](#) = 0x00, [HWDBG\\_BEHAVIOR\\_DISABLE\\_DIS](#)↵  
[PATCH\\_DEBUGGING](#) = 0x01 }
- enum [HwDbgLogType](#) {  
[HWDBG\\_LOG\\_TYPE\\_NONE](#) = 0x00, [HWDBG\\_LOG\\_TYPE\\_ASSERT](#) = 0x01, [HWDBG\\_LOG\\_TYPE\\_ER](#)↵  
[ROR](#) = 0x02, [HWDBG\\_LOG\\_TYPE\\_TRACE](#) = 0x04,  
[HWDBG\\_LOG\\_TYPE\\_MESSAGE](#) = 0x08, [HWDBG\\_LOG\\_TYPE\\_ALL](#) = 0x0f }

## Functions

- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgSetLoggingCallback](#) (uint32\_t types, [Hw](#)↵  
[DbgLoggingCallback](#) pCallback, void \*pUserData)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgGetAPIVersion](#) (uint32\_t \*pVersionMajor↵  
Out, uint32\_t \*pVersionMinorOut, uint32\_t \*pVersionBuildOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgGetAPIType](#) ([HwDbgAPIType](#) \*pAPI↵  
TypeOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgInit](#) (void \*pApiTable)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgShutDown](#) ()
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgBeginDebugContext](#) (const [HwDbgState](#)  
state, [HwDbgContextHandle](#) \*pDebugContextOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgEndDebugContext](#) ([HwDbgContextHandle](#)  
hDebugContext)

- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgWaitForEvent](#) ([HwDbgContextHandle](#) hDebugContext, [const uint32\\_t](#) timeout, [HwDbgEventType](#) \*pEventTypeOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgContinueEvent](#) ([HwDbgContextHandle](#) hDebugContext, [const HwDbgCommand](#) command)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgCreateCodeBreakpoint](#) ([HwDbgContextHandle](#) hDebugContext, [const HwDbgCodeAddress](#) codeAddress, [HwDbgCodeBreakpointHandle](#) \*pBreakpointOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgDeleteCodeBreakpoint](#) ([HwDbgContextHandle](#) hDebugContext, [HwDbgCodeBreakpointHandle](#) hBreakpoint)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgDeleteAllCodeBreakpoints](#) ([HwDbgContextHandle](#) hDebugContext)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgGetCodeBreakpointAddress](#) ([const HwDbgContextHandle](#) hDebugContext, [const HwDbgCodeBreakpointHandle](#) hBreakpoint, [HwDbgCodeAddress](#) \*pCodeAddressOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgGetKernelBinary](#) ([const HwDbgContextHandle](#) hDebugContext, [const void \\*\\*](#)ppBinaryOut, [size\\_t](#) \*pBinarySizeOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgGetDispatchedKernelName](#) ([const HwDbgContextHandle](#) hDebugContext, [const char \\*\\*](#)ppKernelNameOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgGetActiveWavefronts](#) ([const HwDbgContextHandle](#) hDebugContext, [const HwDbgWavefrontInfo \\*\\*](#)ppWavefrontInfoOut, [uint32\\_t](#) \*pNumWavefrontsOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgReadMemory](#) ([const HwDbgContextHandle](#) hDebugContext, [const uint32\\_t](#) memoryRegion, [const HwDbgDim3](#) workGroupId, [const HwDbgDim3](#) workItemId, [const size\\_t](#) offset, [const size\\_t](#) numBytesToRead, [void \\*](#)pMemOut, [size\\_t](#) \*pNumBytesOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgBreakAll](#) ([const HwDbgContextHandle](#) hDebugContext)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgKillAll](#) ([const HwDbgContextHandle](#) hDebugContext)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgCreateDataBreakpoint](#) ([HwDbgContextHandle](#) hDebugContext, [const HwDbgDataBreakpointInfo](#) breakpointInfo, [HwDbgDataBreakpointHandle](#) \*pDataBreakpointOut)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgDeleteDataBreakpoint](#) ([HwDbgContextHandle](#) hDebugContext, [HwDbgDataBreakpointHandle](#) hDataBreakpoint)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgDeleteAllDataBreakpoints](#) ([HwDbgContextHandle](#) hDebugContext)
- [HWDBG\\_API\\_ENTRY HwDbgStatus HWDBG\\_API\\_CALL HwDbgGetDataBreakpointInfo](#) ([const HwDbgContextHandle](#) hDebugContext, [const HwDbgDataBreakpointHandle](#) hDataBreakpoint, [HwDbgDataBreakpointInfo](#) \*pDataBreakpointInfoOut)

### 5.1.1 Detailed Description

The AMD GPU Kernel Debugging API to implement device kernel debugging on AMD Graphics Core Next (GCN) GPUs.

Copyright (c) 2015-2016 Advanced Micro Devices, Inc. All rights reserved.

#### Author

AMD Developer Tools

### 5.1.2 Macro Definition Documentation

#### 5.1.2.1 #define AMDGPUDEBUG\_VERSION\_BUILD 5

The AMD GPU Debug API build number.

Definition at line 83 of file AMDGPUDebug.h.

#### 5.1.2.2 `#define AMDGPUDEBUG_VERSION_MAJOR 1`

The AMD GPU Debug API major version.

Definition at line 79 of file AMDGPUDebug.h.

#### 5.1.2.3 `#define AMDGPUDEBUG_VERSION_MINOR 3`

The AMD GPU Debug API minor version.

Definition at line 81 of file AMDGPUDebug.h.

#### 5.1.2.4 `#define HWDBG_API_CALL`

The API calling convention on linux.

Definition at line 75 of file AMDGPUDebug.h.

#### 5.1.2.5 `#define HWDBG_API_ENTRY`

No symbol visibility control for GCC older than 4.0.

Definition at line 72 of file AMDGPUDebug.h.

#### 5.1.2.6 `#define HWDBG_WAVEFRONT_SIZE 64`

The maximum number of lanes in a wavefront for the GPU device.

Definition at line 86 of file AMDGPUDebug.h.

### 5.1.3 Typedef Documentation

#### 5.1.3.1 `typedef uint64_t HwDbgCodeAddress`

The code location type (in bytes).

Definition at line 201 of file AMDGPUDebug.h.

#### 5.1.3.2 `typedef void* HwDbgCodeBreakpointHandle`

A unique handle for a code breakpoint (returned by `HwDbgCreateCodeBreakpoint`).

Definition at line 210 of file AMDGPUDebug.h.

### 5.1.3.3 typedef void\* HwDbgContextHandle

A unique handle for the kernel debug context (returned by HwDbgBeginDebugContext).

Definition at line 207 of file AMDGPUDebug.h.

### 5.1.3.4 typedef void\* HwDbgDataBreakpointHandle

A unique handle for a data breakpoint (returned by HwDbgCreateDataBreakpoint).

#### Warning

This is not yet supported

Definition at line 214 of file AMDGPUDebug.h.

### 5.1.3.5 typedef void(\* HwDbgLoggingCallback) (void \*pUserData, const HwDbgLogType type, const char \*const pMessage)

The user provided logging callback function to be registered.

This function will be called when the message with the type registered by the user is generated by the library.

#### Parameters

in	<i>pUserData</i>	The pointer specified by the user during registration
in	<i>type</i>	The type of log message being passed back
in	<i>pMessage</i>	The log message being passed back

Definition at line 326 of file AMDGPUDebug.h.

### 5.1.3.6 typedef uint32\_t HwDbgWavefrontAddress

The hardware wavefront location type.

Definition at line 204 of file AMDGPUDebug.h.

## 5.1.4 Enumeration Type Documentation

### 5.1.4.1 enum HwDbgAPIType

The enumeration values of possible driver software stacks supported by the library

#### Enumerator

**HWDBG\_API\_HSA** the library is built for HSA software stack

Definition at line 156 of file AMDGPUDebug.h.

#### 5.1.4.2 enum HwDbgBehaviorType

The enumerated bitfield values of supported behavior, the flags can be used internally to optimize behavior

Enumerator

**HWDBG\_BEHAVIOR\_NONE** Default flag, used to debug GPU dispatches  
**HWDBG\_BEHAVIOR\_DISABLE\_DISPATCH\_DEBUGGING** Disable GPU dispatch debugging. However this behavior mode allows extraction of kernel binaries and breakpoint management. Allowed API calls are HwDbg[Begin or End]DebugContext, HwDbgGetKernelBinary, HwDbg[CodeBreakpoint] and HwDbg[DataBreakpoint]

Definition at line 272 of file AMDGPUDebug.h.

#### 5.1.4.3 enum HwDbgBreakpointType

The enumeration values of possible breakpoint types supported by the library.

Warning

This is not yet supported

Enumerator

**HWDBG\_BREAKPOINT\_TYPE\_NONE** no breakpoint type  
**HWDBG\_BREAKPOINT\_TYPE\_CODE** instruction-based breakpoint type  
**HWDBG\_BREAKPOINT\_TYPE\_DATA** memory-based or data breakpoint type

Definition at line 163 of file AMDGPUDebug.h.

#### 5.1.4.4 enum HwDbgCommand

The list of debugger commands for the HwDbgContinueEvent API to advance to the next state in the GPU debug engine.

Enumerator

**HWDBG\_COMMAND\_CONTINUE** resume the device execution

Definition at line 150 of file AMDGPUDebug.h.

#### 5.1.4.5 enum HwDbgDataBreakpointMode

The list of possible access modes of data breakpoints supported.

Warning

This is not yet supported

Enumerator

**HWDBG\_DATABREAKPOINT\_MODE\_READ** read operations only  
**HWDBG\_DATABREAKPOINT\_MODE\_NONREAD** write or atomic operations only  
**HWDBG\_DATABREAKPOINT\_MODE\_ATOMIC** atomic operations only  
**HWDBG\_DATABREAKPOINT\_MODE\_ALL** read, write or atomic operations

Definition at line 182 of file AMDGPUDebug.h.



## 5.1.4.6 enum HwDbgEventType

The enumeration values of possible event types returned by the HwDbgWaitForEvent API.

Enumerator

**HWDBG\_EVENT\_INVALID** an invalid event  
**HWDBG\_EVENT\_TIMEOUT** has reached the user timeout value  
**HWDBG\_EVENT\_POST\_BREAKPOINT** has reached a breakpoint  
**HWDBG\_EVENT\_END\_DEBUGGING** has completed kernel execution

Definition at line 172 of file AMDGPUDebug.h.

## 5.1.4.7 enum HwDbgLogType

The enumerated bitfield values of supported logging message types

Enumerator

**HWDBG\_LOG\_TYPE\_NONE** do not register for any message  
**HWDBG\_LOG\_TYPE\_ASSERT** register for assert messages  
**HWDBG\_LOG\_TYPE\_ERROR** register for error messages  
**HWDBG\_LOG\_TYPE\_TRACE** register for trace messages  
**HWDBG\_LOG\_TYPE\_MESSAGE** register for generic messages  
**HWDBG\_LOG\_TYPE\_ALL** register for all messages

Definition at line 306 of file AMDGPUDebug.h.

## 5.1.4.8 enum HwDbgStatus

The enumeration values of the possible return status from the provided API.

Warning

Not all the enum values are supported currently

Enumerator

**HWDBG\_STATUS\_SUCCESS** the API was executed successfully  
**HWDBG\_STATUS\_ERROR** a debugger internal error occurred  
**HWDBG\_STATUS\_DEVICE\_ERROR** the GPU device does not support debugging  
**HWDBG\_STATUS\_DRIVER\_ERROR** the driver is not compatible with the API  
**HWDBG\_STATUS\_DUPLICATE\_BREAKPOINT** a duplicate breakpoint is detected  
**HWDBG\_STATUS\_INVALID\_ADDRESS\_ALIGNMENT** invalid address alignment was provided  
**HWDBG\_STATUS\_INVALID\_HANDLE** an invalid debug context handle was provided  
**HWDBG\_STATUS\_INVALID\_PARAMETER** invalid input arguments were provided  
**HWDBG\_STATUS\_NULL\_POINTER** expected a non NULL input argument  
**HWDBG\_STATUS\_OUT\_OF\_RANGE\_ADDRESS** out of range address was provided  
**HWDBG\_STATUS\_OUT\_OF\_MEMORY** failed to allocate memory  
**HWDBG\_STATUS\_OUT\_OF\_RESOURCES** ran out of hardware resources (for data breakpoints)  
**HWDBG\_STATUS\_REGISTRATION\_ERROR** started debugging on more than one application process  
**HWDBG\_STATUS\_UNDEFINED** an undefined operation was detected (i.e. an incorrect call order)  
**HWDBG\_STATUS\_UNSUPPORTED** the API has not been implemented  
**HWDBG\_STATUS\_NOT\_INITIALIZED** HwDbgInit has not been called  
**HWDBG\_STATUS\_INVALID\_BEHAVIOR\_STATE** The debug context was created with unsupported behavior flags for the API

Definition at line 93 of file AMDGPUDebug.h.

### 5.1.5 Function Documentation

#### 5.1.5.1 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgBeginDebugContext ( const HwDbgState state, HwDbgContextHandle \* pDebugContextOut )

Mark the start debugging of a kernel dispatch.

This function should be called right before the execution of the kernel to be debugged (such as within the pre-dispatch callback function). Only one kernel dispatch should be between HwDbgBeginDebugContext and HwDbgEndDebugContext. Only one process can be debugged at a time in the system.

##### Parameters

in	state	specifies the input debug state
out	pDebugContextOut	returns the handle that identifies the particular kernel debug context

##### Returns

HwDbgStatus

##### Return values

HWDBG_STATUS_SUCCESS	On success
HWDBG_STATUS_ERROR	If an internal error occurs (check the log output for details)
HWDBG_STATUS_NOT_INITIALIZED	If called prior to a HwDbgInit call
HWDBG_STATUS_NULL_POINTER	If the input argument is NULL
HWDBG_STATUS_OUT_OF_MEMORY	If fail to allocate necessary memory
HWDBG_STATUS_REGISTRATION_ERROR	If more than 1 debug process is detected
HWDBG_STATUS_UNSUPPORTED	If the API has not been implemented

##### See also

[HwDbgEndDebugContext](#)

#### 5.1.5.2 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgBreakAll ( const HwDbgContextHandle hDebugContext )

Break kernel execution of all active wavefronts for a kernel dispatch.

Can be called at any time after a HwDbgBeginDebugContext call.

##### Parameters

in	hDebugContext	specifies the context handle received from HwDbgBeginDebugContext API
----	---------------	---

##### Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input <i>hDebugContext</i> is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a <i>HwDbgInit</i> call
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

#### 5.1.5.3 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgContinueEvent ( HwDbgContextHandle *hDebugContext*, const HwDbgCommand *command* )

Continue to the next operation (resume device execution, run to the next breakpoint).

This is performed after receiving an event from *HwDbgWaitForEvent*. This is an asynchronous call, subsequent calls are undefined until the next *HwDbgWaitEvent* call.

## Parameters

in	<i>hDebugContext</i>	specifies the context handle received from <i>HwDbgBeginDebugContext</i> API
in	<i>command</i>	specifies the debugger command to execute next

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input <i>hDebugContext</i> is invalid
<i>HWDBG_STATUS_INVALID_PARAMETER</i>	If the command argument is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a <i>HwDbgInit</i> call
<i>HWDBG_STATUS_UNDEFINED</i>	If the kernel has completed execution
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

## See also

[HwDbgWaitForEvent](#)

#### 5.1.5.4 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgCreateCodeBreakpoint ( HwDbgContextHandle *hDebugContext*, const HwDbgCodeAddress *codeAddress*, HwDbgCodeBreakpointHandle \* *pBreakpointOut* )

Create a breakpoint at a specified program counter.

## Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
in	<i>codeAddress</i>	specifies the byte offset into the ISA binary indicating where to set the breakpoint. This has to be 4-byte aligned for AMD GPUs.
out	<i>pBreakpointOut</i>	returns the handle of the newly created instruction-based breakpoint. It is valid for use anywhere after creation. However, it is undefined to change the breakpoint state outside the HwDbgWaitForEvent/ HwDbgContinueEvent pair associated with the kernel dispatch that the breakpoint was created for

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If the codeAddress is invalid (not 4-byte aligned or out of range) or has been inserted before
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_NULL_POINTER</i>	If the input argument is NULL
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

## See also

[HwDbgDeleteCodeBreakpoint](#), [HwDbgDeleteAllCodeBreakpoints](#), [HwDbgGetCodeBreakpointAddress](#)

**5.1.5.5 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgCreateDataBreakpoint**  
 ( HwDbgContextHandle *hDebugContext*, const HwDbgDataBreakpointInfo *breakpointInfo*,  
 HwDbgDataBreakpointHandle \* *pDataBreakpointOut* )

Create a data breakpoint.

## Warning

This is not yet supported

## Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
in	<i>breakpointInfo</i>	specifies the structure containing information where to set the data breakpoint
out	<i>pDataBreakpointOut</i>	returns the handle of the newly created data breakpoint. It is valid for use anywhere after creation. However, it is undefined to change the breakpoint state outside the HwDbgWaitForEvent/ HwDbgContinueEvent pair associated with the shader dispatch that the breakpoint was created for

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_NULL_POINTER</i>	If the input argument or address is NULL
<i>HWDBG_STATUS_OUT_OF_RESOURCES</i>	If cannot be created due to hw limits
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

## See also

[HwDbgDeleteDataBreakpoint](#), [HwDbgDeleteAllDataBreakpoints](#), [HwDbgGetDataBreakpointInfo](#)

#### 5.1.5.6 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgDeleteAllCodeBreakpoints ( HwDbgContextHandle hDebugContext )

Delete all instruction-based breakpoints.

## Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
----	----------------------	---

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

## See also

[HwDbgCreateCodeBreakpoint](#), [HwDbgDeleteCodeBreakpoint](#), [HwDbgGetCodeBreakpointAddress](#)

#### 5.1.5.7 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgDeleteAllDataBreakpoints ( HwDbgContextHandle *hDebugContext* )

Delete all data breakpoints.

##### Warning

This is not yet supported

After this call, all data breakpoint handles created prior for the debug context will be invalid.

##### Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
----	----------------------	---

##### Returns

HwDbgStatus

##### Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input <i>hDebugContext</i> is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

##### See also

[HwDbgCreateDataBreakpoint](#), [HwDbgDeleteDataBreakpoint](#), [HwDbgGetDataBreakpointInfo](#)

#### 5.1.5.8 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgDeleteCodeBreakpoint ( HwDbgContextHandle *hDebugContext*, HwDbgCodeBreakpointHandle *hBreakpoint* )

Delete a instruction-based breakpoint.

##### Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
in	<i>hBreakpoint</i>	specifies the breakpoint handle. The handle is invalid after this call and may be returned in future calls to HwDbgCreateCodeBreakpoint

##### Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If breakpoint handle is invalid or contains an invalid code address
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

## See also

[HwDbgCreateCodeBreakpoint](#), [HwDbgDeleteAllCodeBreakpoints](#), [HwDbgGetCodeBreakpointAddress](#)

#### 5.1.5.9 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgDeleteDataBreakpoint ( HwDbgContextHandle hDebugContext, HwDbgDataBreakpointHandle hDataBreakpoint )

Delete a data breakpoint.

## Warning

This is not yet supported

## Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
in	<i>hDataBreakpoint</i>	specifies the data breakpoint handle. The handle is invalid after this call and may be returned in future calls to HwDbgCreateCodeBreakpoint

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

## See also

[HwDbgCreateDataBreakpoint](#), [HwDbgDeleteAllDataBreakpoints](#), [HwDbgGetDataBreakpointInfo](#)

#### 5.1.5.10 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgEndDebugContext ( HwDbgContextHandle *hDebugContext* )

Mark the end debugging of a kernel dispatch.

This function must be called after the kernel has complete execution. Only one kernel dispatch should be between HwDbgBeginDebugContext and HwDbgEndDebugContext. Only one process can be debugged at a time in the system.

##### Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API. If it is NULL, then all sessions in flight will be terminated and deleted
----	----------------------	--

##### Returns

HwDbgStatus

##### Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If <i>hDebugContext</i> is an invalid handle
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_UNDEFINED</i>	If kernel execution has not yet completed
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

##### See also

[HwDbgBeginDebugContext](#)

#### 5.1.5.11 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgGetActiveWavefronts ( const HwDbgContextHandle *hDebugContext*, const HwDbgWavefrontInfo \*\* *ppWavefrontInfoOut*, uint32\_t \* *pNumWavefrontsOut* )

Retrieve the list of active wavefronts for the kernel dispatch in the GPU device.

Must only be called after receiving a HWDBG\_EVENT\_POST\_BREAKPOINT event from HwDbgWaitForEvent API.

##### Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
out	<i>ppWavefrontInfoOut</i>	returns a pointer to <a href="#">HwDbgWavefrontInfo</a> structures. It contains the work-group ids, work- item ids, code address, etc for each wavefront
out	<i>pNumWavefrontsOut</i>	returns the number of active wavefronts

##### Returns

HwDbgStatus



## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_NULL_POINTER</i>	If the ppWaveInfoOut is NULL
<i>HWDBG_STATUS_UNDEFINED</i>	If it is called after not receiving a HWDBG_EVENT_POST_BREAKPOINT event
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

#### 5.1.5.12 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgGetAPIType ( HwDbgAPIType \* pAPITypeOut )

Retrieve the driver API type of the loaded library.

This function can be called prior to a HwDbgInit call.

## Parameters

out	<i>pAPITypeOut</i>	returns the API type of the library
-----	--------------------	-------------------------------------

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_NULL_POINTER</i>	If the input argument is NULL
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API is not yet implemented

#### 5.1.5.13 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgGetAPIVersion ( uint32\_t \* pVersionMajorOut, uint32\_t \* pVersionMinorOut, uint32\_t \* pVersionBuildOut )

Retrieve the library version (major, minor and build) number.

This function can be called prior to a HwDbgInit call.

## Parameters

out	<i>pVersionMajorOut</i>	returns the API version major number
out	<i>pVersionMinorOut</i>	returns API version minor number
out	<i>pVersionBuildOut</i>	returns API build number

**Returns**

HwDbgStatus

**Return values**

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_NULL_POINTER</i>	If an input argument is NULL
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API is not yet implemented

**5.1.5.14 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgGetCodeBreakpointAddress (**  
**const HwDbgContextHandle *hDebugContext*, const HwDbgCodeBreakpointHandle *hBreakpoint*,**  
**HwDbgCodeAddress \* *pCodeAddressOut* )**

Retrieve the code location from an instruction-based breakpoint handle.

**Parameters**

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
in	<i>hBreakpoint</i>	specifies the breakpoint handle
out	<i>pCodeAddressOut</i>	returns the code address (program counter)

**Returns**

HwDbgStatus

HWDBG\_STATUS\_SUCCESS On success

**Return values**

<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input <i>hDebugContext</i> is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_NULL_POINTER</i>	If the input argument is NULL
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

**See also**

[HwDbgCreateCodeBreakpoint](#), [HwDbgDeleteCodeBreakpoint](#), [HwDbgDeleteAllCodeBreakpoints](#)

**5.1.5.15 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgGetDataBreakpointInfo (**  
**const HwDbgContextHandle *hDebugContext*, const HwDbgDataBreakpointHandle *hDataBreakpoint*,**  
**HwDbgDataBreakpointInfo \* *pDataBreakpointInfoOut* )**

Retrieve the data breakpoint information from a data breakpoint handle.

**Warning**

This is not yet supported

## Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
in	<i>hDataBreakpoint</i>	specifies the data breakpoint handle
out	<i>pDataBreakpointInfoOut</i>	returns a structure containing information of the data breakpoint

## Return values

<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
-------------------------------------	-------------------------------------

## Returns

HwDbgStatus  
HWDBG\_STATUS\_SUCCESS On success

## Return values

<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NULL_POINTER</i>	If the input argument is NULL
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

## See also

[HwDbgCreateDataBreakpoint](#), [HwDbgDeleteDataBreakpoint](#), [HwDbgDeleteAllDataBreakpoints](#)

#### 5.1.5.16 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgGetDispatchedKernelName ( const HwDbgContextHandle *hDebugContext*, const char \*\* *ppKernelNameOut* )

Retrieve the dispatched kernel name.

## Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
out	<i>ppKernelNameOut</i>	returns a pointer to a null-terminated character array The lifetime of the character array is within the debug context (i.e. after HwDbgBeginDebugContext call until the HwDbgEndDebugContext call)

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_DRIVER_ERROR</i>	If the retrieved kernel name is NULL

## Return values

<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input <i>hDebugContext</i> is invalid
<i>HWDBG_STATUS_NULL_POINTER</i>	If the input argument is NULL
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a <i>HwDbgInit</i> call
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

#### 5.1.5.17 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgGetKernelBinary ( const HwDbgContextHandle *hDebugContext*, const void \*\* *ppBinaryOut*, size\_t \* *pBinarySizeOut* )

Retrieve the kernel binary (in ELF) of the kernel dispatch.

For HSA, the binary is the loaded and relocated code object. The binary contains the debugging information (in DWARF) from high level source to ISA (can be multiple level of DWARFs such as one DWARF to represent the mapping from a high level kernel source to BRIG and another DWARF to represent the mapping from BRIG to ISA).

## Note

Refer to the following two documentation for more information:

1. HSA Application Binary Interface - AMD GPU Architecture document for the complete ABI.
2. HSA Debug Information document for the HSA DWARF extension

## Parameters

in	<i>hDebugContext</i>	specifies the context handle received from <i>HwDbgBeginDebugContext</i> API
out	<i>ppBinaryOut</i>	returns a pointer to a buffer containing the binary kernel code object The lifetime of the buffer is within the debug context (i.e. after <i>HwDbgBeginDebugContext</i> call until the <i>HwDbgEndDebugContext</i> call)
out	<i>pBinarySizeOut</i>	returns the binary size in bytes

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_DRIVER_ERROR</i>	If the retrieved kernel binary is NULL or the binary size is 0
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input <i>hDebugContext</i> is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a <i>HwDbgInit</i> call
<i>HWDBG_STATUS_NULL_POINTER</i>	If the input argument is NULL
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

**5.1.5.18 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgInit ( void \* *pApiTable* )**

Initialize the GPU debug engine.

This function should be called right after the debugged process starts. For hsa, this is in the HSA Runtime's OnLoad callback.

**Parameters**

in	<i>pApiTable</i>	Used by HSA: Pass in the pointer to the hsa api table provided by the HSA Runtime's OnLoad callback. Can be NULL (won't support full DBE functionality).
----	------------------	--

**Returns**

HwDbgStatus

**Return values**

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If called multiple times without a corresponding HwDbgShutDown
<i>HWDBG_STATUS_OUT_OF_MEMORY</i>	If fail to allocate necessary memory

**See also**

[HwDbgShutDown](#)

**5.1.5.19 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgKillAll ( const HwDbgContextHandle *hDebugContext* )**

Terminate the kernel dispatch execution.

Can be called at any time after a HwDbgBeginDebugContext call. Can be called multiple times to terminate a large kernel dispatch.

**Parameters**

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
----	----------------------	---

**Returns**

HwDbgStatus

**Return values**

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

**5.1.5.20 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgReadMemory ( const HwDbgContextHandle *hDebugContext*, const uint32\_t *memoryRegion*, const HwDbgDim3 *workGroupId*, const HwDbgDim3 *workItemId*, const size\_t *offset*, const size\_t *numBytesToRead*, void \* *pMemOut*, size\_t \* *pNumBytesOut* )**

Read data from a memory region.

#### Warning

Only private memory region (IMR\_Scratch = 1) is currently supported.

Must only be called after receiving a HWDBG\_EVENT\_POST\_BREAKPOINT event from HwDbgWaitForEvent API.

#### Parameters

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
in	<i>memoryRegion</i>	specifies the target memory region to read from. This should be set to an enum value stored in DW_AT_HSA_isa_memory_region attribute of DW_TAG_variable tag in ISA DWARF.
in	<i>workGroupId</i>	specifies the work-group id of interest (from HwDbgGetActiveWavefronts)
in	<i>workItemId</i>	specifies the work-item id of interest (from HwDbgGetActiveWavefronts)
in	<i>offset</i>	specifies a byte offset for the logical location that should be retrieved. On GPU, this must be a multiple of 4 bytes (align on a DWORD boundary)
in	<i>numBytesToRead</i>	specifies the number of bytes to retrieve On GPU, this must be a multiple of 4 bytes
out	<i>pMemOut</i>	returns a pointer to a memory chunk of at least "numBytesToRead" bytes long
out	<i>pNumBytesOut</i>	returns the number of bytes written into pMemOut

#### Returns

HwDbgStatus

#### Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input <i>hDebugContext</i> is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_NULL_POINTER</i>	If an input argument is NULL
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

**5.1.5.21 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgSetLoggingCallback ( uint32\_t *types*, HwDbgLoggingCallback *pCallback*, void \* *pUserData* )**

Register a logging callback function.

Extra diagnostics output about the operation of the AMD GPU Debug API may be enabled by registering a client callback function through this API.

This function can be called prior to a HwDbgInit call.

## Parameters

in	<i>types</i>	specifies the logging message types to be registered (a combination of HwDbgLogType enum value)
in	<i>pCallback</i>	specifies the logging callback function Set to a callback function function to enable logging Set to NULL to disable logging
in	<i>pUserData</i>	specifies a pointer to data that can be accessed by the user specified logging callback function

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	If the callback can be set successfully
<i>HWDBG_STATUS_ERROR</i>	If an error is encountered
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

## 5.1.5.22 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgShutDown ( )

Shut down the GPU debug engine.

This function should be called before the debugged process ends. For hsa, this should be called right before calling the hsa\_shut\_down API.

## Returns

HwDbgStatus

## Return values

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called without a corresponding HwDbgInit

## See also

[HwDbgInit](#)

## 5.1.5.23 HWDBG\_API\_ENTRY HwDbgStatus HWDBG\_API\_CALL HwDbgWaitForEvent ( HwDbgContextHandle hDebugContext, const uint32\_t timeout, HwDbgEventType \* pEventTypeOut )

Wait on a debug event from the GPU device.

This is a synchronous function that will not return until it receives an event or reaches the specified timeout value.

**Parameters**

in	<i>hDebugContext</i>	specifies the context handle received from HwDbgBeginDebugContext API
in	<i>timeout</i>	specifies how long to wait in milliseconds before timing out
out	<i>pEventTypeOut</i>	The resulting event type

**Returns**

HwDbgStatus

**Return values**

<i>HWDBG_STATUS_SUCCESS</i>	On success
<i>HWDBG_STATUS_ERROR</i>	If an internal error occurs (check the log output for details)
<i>HWDBG_STATUS_INVALID_BEHAVIOR</i>	If the context behavior flags are invalid
<i>HWDBG_STATUS_INVALID_HANDLE</i>	If the input hDebugContext is invalid
<i>HWDBG_STATUS_NOT_INITIALIZED</i>	If called prior to a HwDbgInit call
<i>HWDBG_STATUS_NULL_POINTER</i>	If an input argument is NULL
<i>HWDBG_STATUS_UNDEFINED</i>	If the kernel has completed execution
<i>HWDBG_STATUS_UNSUPPORTED</i>	If the API has not been implemented

**See also**

[HwDbgContinueEvent](#)



# Index

- AMDGPUDEBUG\_VERSION\_BUILD
  - AMDGPUDebug.h, 15
- AMDGPUDEBUG\_VERSION\_MAJOR
  - AMDGPUDebug.h, 15
- AMDGPUDEBUG\_VERSION\_MINOR
  - AMDGPUDebug.h, 16
- AMDGPUDebug.h, 13
  - AMDGPUDEBUG\_VERSION\_BUILD, 15
  - AMDGPUDEBUG\_VERSION\_MAJOR, 15
  - AMDGPUDEBUG\_VERSION\_MINOR, 16
  - HWDBG\_API\_CALL, 16
  - HWDBG\_API\_ENTRY, 16
  - HWDBG\_API\_HSA, 17
  - HWDBG\_BEHAVIOR\_DISABLE\_DISPATCH\_D←  
EBUGGING, 18
  - HWDBG\_BEHAVIOR\_NONE, 18
  - HWDBG\_BREAKPOINT\_TYPE\_CODE, 18
  - HWDBG\_BREAKPOINT\_TYPE\_DATA, 18
  - HWDBG\_BREAKPOINT\_TYPE\_NONE, 18
  - HWDBG\_COMMAND\_CONTINUE, 18
  - HWDBG\_DATABREAKPOINT\_MODE\_ALL, 18
  - HWDBG\_DATABREAKPOINT\_MODE\_ATOMIC,  
18
  - HWDBG\_DATABREAKPOINT\_MODE\_NONRE←  
AD, 18
  - HWDBG\_DATABREAKPOINT\_MODE\_READ, 18
  - HWDBG\_EVENT\_END\_DEBUGGING, 19
  - HWDBG\_EVENT\_INVALID, 19
  - HWDBG\_EVENT\_POST\_BREAKPOINT, 19
  - HWDBG\_EVENT\_TIMEOUT, 19
  - HWDBG\_LOG\_TYPE\_ALL, 19
  - HWDBG\_LOG\_TYPE\_ASSERT, 19
  - HWDBG\_LOG\_TYPE\_ERROR, 19
  - HWDBG\_LOG\_TYPE\_MESSAGE, 19
  - HWDBG\_LOG\_TYPE\_NONE, 19
  - HWDBG\_LOG\_TYPE\_TRACE, 19
  - HWDBG\_STATUS\_DEVICE\_ERROR, 19
  - HWDBG\_STATUS\_DRIVER\_ERROR, 19
  - HWDBG\_STATUS\_DUPLICATE\_BREAKPOINT,  
19
  - HWDBG\_STATUS\_ERROR, 19
  - HWDBG\_STATUS\_INVALID\_ADDRESS\_ALIG←  
NMENT, 19
  - HWDBG\_STATUS\_INVALID\_BEHAVIOR\_STA←  
TE, 19
  - HWDBG\_STATUS\_INVALID\_HANDLE, 19
  - HWDBG\_STATUS\_INVALID\_PARAMETER, 19
  - HWDBG\_STATUS\_NOT\_INITIALIZED, 19
  - HWDBG\_STATUS\_NULL\_POINTER, 19
  - HWDBG\_STATUS\_OUT\_OF\_MEMORY, 19
  - HWDBG\_STATUS\_OUT\_OF\_RANGE\_ADDRE←  
SS, 19
  - HWDBG\_STATUS\_OUT\_OF\_RESOURCES, 19
  - HWDBG\_STATUS\_REGISTRATION\_ERROR, 19
  - HWDBG\_STATUS\_SUCCESS, 19
  - HWDBG\_STATUS\_UNDEFINED, 19
  - HWDBG\_STATUS\_UNSUPPORTED, 19
  - HWDBG\_WAVEFRONT\_SIZE, 16
  - HwDbgAPIType, 17
  - HwDbgBeginDebugContext, 20
  - HwDbgBehaviorType, 17
  - HwDbgBreakAll, 20
  - HwDbgBreakpointType, 18
  - HwDbgCodeAddress, 16
  - HwDbgCodeBreakpointHandle, 16
  - HwDbgCommand, 18
  - HwDbgContextHandle, 16
  - HwDbgContinueEvent, 21
  - HwDbgCreateCodeBreakpoint, 21
  - HwDbgCreateDataBreakpoint, 22
  - HwDbgDataBreakpointHandle, 17
  - HwDbgDataBreakpointMode, 18
  - HwDbgDeleteAllCodeBreakpoints, 23
  - HwDbgDeleteAllDataBreakpoints, 23
  - HwDbgDeleteCodeBreakpoint, 24
  - HwDbgDeleteDataBreakpoint, 25
  - HwDbgEndDebugContext, 25
  - HwDbgEventType, 18
  - HwDbgGetAPIType, 27
  - HwDbgGetAPIVersion, 27
  - HwDbgGetActiveWavefronts, 26
  - HwDbgGetCodeBreakpointAddress, 28
  - HwDbgGetDataBreakpointInfo, 28
  - HwDbgGetDispatchedKernelName, 29
  - HwDbgGetKernelBinary, 30
  - HwDbgInit, 30
  - HwDbgKillAll, 31
  - HwDbgLogType, 19
  - HwDbgLoggingCallback, 17
  - HwDbgReadMemory, 32
  - HwDbgSetLoggingCallback, 32
  - HwDbgShutDown, 33
  - HwDbgStatus, 19
  - HwDbgWaitForEvent, 33
  - HwDbgWavefrontAddress, 17
- behaviorFlags
  - HwDbgState, 9
- breakpointType

- HwDbgWavefrontInfo, [10](#)
- codeAddress
  - HwDbgWavefrontInfo, [10](#)
- dataBreakpointHandle
  - HwDbgWavefrontInfo, [11](#)
- dataBreakpointMode
  - HwDbgDataBreakpointInfo, [7](#)
- dataSize
  - HwDbgDataBreakpointInfo, [7](#)
- executionMask
  - HwDbgWavefrontInfo, [11](#)
- HWDBG\_API\_CALL
  - AMDGPUDebug.h, [16](#)
- HWDBG\_API\_ENTRY
  - AMDGPUDebug.h, [16](#)
- HWDBG\_API\_HSA
  - AMDGPUDebug.h, [17](#)
- HWDBG\_BEHAVIOR\_DISABLE\_DISPATCH\_DEBUGGING
  - AMDGPUDebug.h, [18](#)
- HWDBG\_BEHAVIOR\_NONE
  - AMDGPUDebug.h, [18](#)
- HWDBG\_BREAKPOINT\_TYPE\_CODE
  - AMDGPUDebug.h, [18](#)
- HWDBG\_BREAKPOINT\_TYPE\_DATA
  - AMDGPUDebug.h, [18](#)
- HWDBG\_BREAKPOINT\_TYPE\_NONE
  - AMDGPUDebug.h, [18](#)
- HWDBG\_COMMAND\_CONTINUE
  - AMDGPUDebug.h, [18](#)
- HWDBG\_DATABREAKPOINT\_MODE\_ALL
  - AMDGPUDebug.h, [18](#)
- HWDBG\_DATABREAKPOINT\_MODE\_ATOMIC
  - AMDGPUDebug.h, [18](#)
- HWDBG\_DATABREAKPOINT\_MODE\_NONREAD
  - AMDGPUDebug.h, [18](#)
- HWDBG\_DATABREAKPOINT\_MODE\_READ
  - AMDGPUDebug.h, [18](#)
- HWDBG\_EVENT\_END\_DEBUGGING
  - AMDGPUDebug.h, [19](#)
- HWDBG\_EVENT\_INVALID
  - AMDGPUDebug.h, [19](#)
- HWDBG\_EVENT\_POST\_BREAKPOINT
  - AMDGPUDebug.h, [19](#)
- HWDBG\_EVENT\_TIMEOUT
  - AMDGPUDebug.h, [19](#)
- HWDBG\_LOG\_TYPE\_ALL
  - AMDGPUDebug.h, [19](#)
- HWDBG\_LOG\_TYPE\_ASSERT
  - AMDGPUDebug.h, [19](#)
- HWDBG\_LOG\_TYPE\_ERROR
  - AMDGPUDebug.h, [19](#)
- HWDBG\_LOG\_TYPE\_MESSAGE
  - AMDGPUDebug.h, [19](#)
- HWDBG\_LOG\_TYPE\_NONE
  - AMDGPUDebug.h, [19](#)
- HWDBG\_LOG\_TYPE\_TRACE
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_DEVICE\_ERROR
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_DRIVER\_ERROR
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_DUPLICATE\_BREAKPOINT
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_ERROR
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_INVALID\_ADDRESS\_ALIGNMENT
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_INVALID\_BEHAVIOR\_STATE
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_INVALID\_HANDLE
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_INVALID\_PARAMETER
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_NOT\_INITIALIZED
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_NULL\_POINTER
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_OUT\_OF\_MEMORY
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_OUT\_OF\_RANGE\_ADDRESS
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_OUT\_OF\_RESOURCES
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_REGISTRATION\_ERROR
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_SUCCESS
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_UNDEFINED
  - AMDGPUDebug.h, [19](#)
- HWDBG\_STATUS\_UNSUPPORTED
  - AMDGPUDebug.h, [19](#)
- HWDBG\_WAVEFRONT\_SIZE
  - AMDGPUDebug.h, [16](#)
- HwDbgAPIType
  - AMDGPUDebug.h, [17](#)
- HwDbgBeginDebugContext
  - AMDGPUDebug.h, [20](#)
- HwDbgBehaviorType
  - AMDGPUDebug.h, [17](#)
- HwDbgBreakAll
  - AMDGPUDebug.h, [20](#)
- HwDbgBreakpointType
  - AMDGPUDebug.h, [18](#)
- HwDbgCodeAddress
  - AMDGPUDebug.h, [16](#)
- HwDbgCodeBreakpointHandle
  - AMDGPUDebug.h, [16](#)
- HwDbgCommand
  - AMDGPUDebug.h, [18](#)
- HwDbgContextHandle
  - AMDGPUDebug.h, [16](#)
- HwDbgContinueEvent

- AMDGPUDebug.h, [21](#)
- HwDbgCreateCodeBreakpoint
  - AMDGPUDebug.h, [21](#)
- HwDbgCreateDataBreakpoint
  - AMDGPUDebug.h, [22](#)
- HwDbgDataBreakpointHandle
  - AMDGPUDebug.h, [17](#)
- HwDbgDataBreakpointInfo, [7](#)
  - dataBreakpointMode, [7](#)
  - dataSize, [7](#)
  - pAddress, [7](#)
- HwDbgDataBreakpointMode
  - AMDGPUDebug.h, [18](#)
- HwDbgDeleteAllCodeBreakpoints
  - AMDGPUDebug.h, [23](#)
- HwDbgDeleteAllDataBreakpoints
  - AMDGPUDebug.h, [23](#)
- HwDbgDeleteCodeBreakpoint
  - AMDGPUDebug.h, [24](#)
- HwDbgDeleteDataBreakpoint
  - AMDGPUDebug.h, [25](#)
- HwDbgDim3, [8](#)
  - x, [8](#)
  - y, [8](#)
  - z, [8](#)
- HwDbgEndDebugContext
  - AMDGPUDebug.h, [25](#)
- HwDbgEventType
  - AMDGPUDebug.h, [18](#)
- HwDbgGetAPIType
  - AMDGPUDebug.h, [27](#)
- HwDbgGetAPIVersion
  - AMDGPUDebug.h, [27](#)
- HwDbgGetActiveWavefronts
  - AMDGPUDebug.h, [26](#)
- HwDbgGetCodeBreakpointAddress
  - AMDGPUDebug.h, [28](#)
- HwDbgGetDataBreakpointInfo
  - AMDGPUDebug.h, [28](#)
- HwDbgGetDispatchedKernelName
  - AMDGPUDebug.h, [29](#)
- HwDbgGetKernelBinary
  - AMDGPUDebug.h, [30](#)
- HwDbgInit
  - AMDGPUDebug.h, [30](#)
- HwDbgKillAll
  - AMDGPUDebug.h, [31](#)
- HwDbgLogType
  - AMDGPUDebug.h, [19](#)
- HwDbgLoggingCallback
  - AMDGPUDebug.h, [17](#)
- HwDbgReadMemory
  - AMDGPUDebug.h, [32](#)
- HwDbgSetLoggingCallback
  - AMDGPUDebug.h, [32](#)
- HwDbgShutDown
  - AMDGPUDebug.h, [33](#)
- HwDbgState, [9](#)
- behaviorFlags, [9](#)
- pDevice, [9](#)
- pPacket, [9](#)
- packetId, [9](#)
- HwDbgStatus
  - AMDGPUDebug.h, [19](#)
- HwDbgWaitForEvent
  - AMDGPUDebug.h, [33](#)
- HwDbgWavefrontAddress
  - AMDGPUDebug.h, [17](#)
- HwDbgWavefrontInfo, [10](#)
  - breakpointType, [10](#)
  - codeAddress, [10](#)
  - dataBreakpointHandle, [11](#)
  - executionMask, [11](#)
  - pOtherData, [11](#)
  - wavefrontAddress, [11](#)
  - workGroupId, [11](#)
  - workItemId, [11](#)
- pAddress
  - HwDbgDataBreakpointInfo, [7](#)
- pDevice
  - HwDbgState, [9](#)
- pOtherData
  - HwDbgWavefrontInfo, [11](#)
- pPacket
  - HwDbgState, [9](#)
- packetId
  - HwDbgState, [9](#)
- wavefrontAddress
  - HwDbgWavefrontInfo, [11](#)
- workGroupId
  - HwDbgWavefrontInfo, [11](#)
- workItemId
  - HwDbgWavefrontInfo, [11](#)
- x
  - HwDbgDim3, [8](#)
- y
  - HwDbgDim3, [8](#)
- z
  - HwDbgDim3, [8](#)