

# Reference Manual

## 0.1.17

Generated by Doxygen 1.8.5

Wed Feb 19 2020 04:32:00



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Module Documentation</b>	<b>5</b>
2.1	Basic Types	5
2.1.1	Detailed Description	5
2.1.2	Typedef Documentation	5
2.1.2.1	amd_dbgapi_global_address_t	5
2.1.2.2	amd_dbgapi_notifier_t	6
2.1.2.3	amd_dbgapi_os_pid	6
2.1.2.4	amd_dbgapi_size_t	6
2.1.3	Enumeration Type Documentation	6
2.1.3.1	amd_dbgapi_changed_t	6
2.2	Status Codes	7
2.2.1	Detailed Description	7
2.2.2	Enumeration Type Documentation	8
2.2.2.1	amd_dbgapi_status_t	8
2.2.3	Function Documentation	9
2.2.3.1	amd_dbgapi_get_status_string	9
2.3	Versioning	10
2.3.1	Detailed Description	10
2.3.2	Macro Definition Documentation	10
2.3.2.1	AMD_DBGAPI_SYMBOL_VERSION	10
2.3.3	Enumeration Type Documentation	10
2.3.3.1	anonymous enum	10
2.3.4	Function Documentation	11
2.3.4.1	amd_dbgapi_get_build_name	11
2.3.4.2	amd_dbgapi_get_version	11
2.4	Initialization and Finalization	12
2.4.1	Detailed Description	12

2.4.2	Function Documentation	12
2.4.2.1	amd_dbgapi_finalize	12
2.4.2.2	amd_dbgapi_initialize	12
2.5	Architectures	14
2.5.1	Detailed Description	14
2.5.2	Macro Definition Documentation	15
2.5.2.1	AMD_DBGAPI_ARCHITECTURE_NONE	15
2.5.3	Enumeration Type Documentation	15
2.5.3.1	amd_dbgapi_architecture_info_t	15
2.5.4	Function Documentation	16
2.5.4.1	amd_dbgapi_architecture_get_info	16
2.5.4.2	amd_dbgapi_disassemble_instruction	16
2.5.4.3	amd_dbgapi_get_architecture	18
2.6	Processes	19
2.6.1	Detailed Description	19
2.6.2	Macro Definition Documentation	20
2.6.2.1	AMD_DBGAPI_PROCESS_NONE	20
2.6.3	Typedef Documentation	20
2.6.3.1	amd_dbgapi_client_process_id_t	20
2.6.4	Enumeration Type Documentation	20
2.6.4.1	amd_dbgapi_process_info_t	20
2.6.4.2	amd_dbgapi_progress_t	20
2.6.5	Function Documentation	21
2.6.5.1	amd_dbgapi_process_attach	21
2.6.5.2	amd_dbgapi_process_detach	22
2.6.5.3	amd_dbgapi_process_get_info	22
2.6.5.4	amd_dbgapi_process_set_progress	23
2.7	Code Objects	24
2.7.1	Detailed Description	24
2.7.2	Macro Definition Documentation	24
2.7.2.1	AMD_DBGAPI_CODE_OBJECT_NONE	24
2.7.3	Enumeration Type Documentation	25
2.7.3.1	amd_dbgapi_code_object_info_t	25
2.7.4	Function Documentation	25
2.7.4.1	amd_dbgapi_code_object_get_info	25
2.7.4.2	amd_dbgapi_code_object_list	26
2.8	Agents	28

2.8.1	Detailed Description	28
2.8.2	Macro Definition Documentation	28
2.8.2.1	AMD_DBGAPI_AGENT_NONE	28
2.8.3	Enumeration Type Documentation	29
2.8.3.1	amd_dbgapi_agent_info_t	29
2.8.4	Function Documentation	29
2.8.4.1	amd_dbgapi_agent_get_info	29
2.8.4.2	amd_dbgapi_agent_list	30
2.9	Queues	32
2.9.1	Detailed Description	33
2.9.2	Macro Definition Documentation	33
2.9.2.1	AMD_DBGAPI_QUEUE_NONE	33
2.9.3	Typedef Documentation	33
2.9.3.1	amd_dbgapi_queue_packet_id_t	33
2.9.4	Enumeration Type Documentation	33
2.9.4.1	amd_dbgapi_queue_error_reason_t	33
2.9.4.2	amd_dbgapi_queue_info_t	33
2.9.4.3	amd_dbgapi_queue_state_t	34
2.9.4.4	amd_dbgapi_queue_type_t	34
2.9.5	Function Documentation	34
2.9.5.1	amd_dbgapi_queue_get_info	35
2.9.5.2	amd_dbgapi_queue_list	36
2.9.5.3	amd_dbgapi_queue_packet_list	37
2.10	Dispatches	39
2.10.1	Detailed Description	39
2.10.2	Macro Definition Documentation	40
2.10.2.1	AMD_DBGAPI_DISPATCH_NONE	40
2.10.3	Enumeration Type Documentation	40
2.10.3.1	amd_dbgapi_dispatch_barrier_t	40
2.10.3.2	amd_dbgapi_dispatch_fence_scope_t	40
2.10.3.3	amd_dbgapi_dispatch_info_t	40
2.10.4	Function Documentation	41
2.10.4.1	amd_dbgapi_dispatch_get_info	41
2.10.4.2	amd_dbgapi_dispatch_list	42
2.11	Wave	43
2.11.1	Detailed Description	44
2.11.2	Macro Definition Documentation	44

2.11.2.1	AMD_DBGAPI_WAVE_NONE	44
2.11.3	Enumeration Type Documentation	44
2.11.3.1	amd_dbgapi_resume_mode_t	44
2.11.3.2	amd_dbgapi_wave_info_t	44
2.11.3.3	amd_dbgapi_wave_state_t	45
2.11.3.4	amd_dbgapi_wave_stop_reason_t	45
2.11.4	Function Documentation	47
2.11.4.1	amd_dbgapi_wave_get_info	47
2.11.4.2	amd_dbgapi_wave_list	48
2.11.4.3	amd_dbgapi_wave_resume	49
2.11.4.4	amd_dbgapi_wave_stop	50
2.12	Displaced Stepping	52
2.12.1	Detailed Description	52
2.12.2	Macro Definition Documentation	53
2.12.2.1	AMD_DBGAPI_DISPLACED_STEPPING_NONE	53
2.12.3	Function Documentation	53
2.12.3.1	amd_dbgapi_displaced_stepping_complete	53
2.12.3.2	amd_dbgapi_displaced_stepping_start	54
2.13	Watchpoints	56
2.13.1	Detailed Description	56
2.13.2	Macro Definition Documentation	57
2.13.2.1	AMD_DBGAPI_WATCHPOINT_NONE	57
2.13.3	Typedef Documentation	57
2.13.3.1	amd_dbgapi_watchpoint_id_t	57
2.13.4	Enumeration Type Documentation	57
2.13.4.1	amd_dbgapi_watchpoint_kind_t	57
2.13.4.2	amd_dbgapi_watchpoint_share_kind_t	57
2.13.5	Function Documentation	58
2.13.5.1	amd_dbgapi_remove_watchpoint	58
2.13.5.2	amd_dbgapi_set_watchpoint	59
2.14	Registers	61
2.14.1	Detailed Description	62
2.14.2	Macro Definition Documentation	62
2.14.2.1	AMD_DBGAPI_REGISTER_CLASS_NONE	62
2.14.2.2	AMD_DBGAPI_REGISTER_NONE	62
2.14.3	Enumeration Type Documentation	62
2.14.3.1	amd_dbgapi_register_class_info_t	62

2.14.3.2	<a href="#">amd_dbgapi_register_class_state_t</a>	63
2.14.3.3	<a href="#">amd_dbgapi_register_info_t</a>	63
2.14.4	Function Documentation	63
2.14.4.1	<a href="#">amd_dbgapi_architecture_register_class_get_info</a>	64
2.14.4.2	<a href="#">amd_dbgapi_architecture_register_class_list</a>	64
2.14.4.3	<a href="#">amd_dbgapi_architecture_register_get_info</a>	65
2.14.4.4	<a href="#">amd_dbgapi_architecture_register_list</a>	66
2.14.4.5	<a href="#">amd_dbgapi_dwarf_register_to_register</a>	67
2.14.4.6	<a href="#">amd_dbgapi_prefetch_register</a>	67
2.14.4.7	<a href="#">amd_dbgapi_read_register</a>	68
2.14.4.8	<a href="#">amd_dbgapi_register_is_in_register_class</a>	69
2.14.4.9	<a href="#">amd_dbgapi_wave_register_get_info</a>	70
2.14.4.10	<a href="#">amd_dbgapi_wave_register_list</a>	71
2.14.4.11	<a href="#">amd_dbgapi_write_register</a>	72
2.15	Memory	73
2.15.1	Detailed Description	75
2.15.2	Macro Definition Documentation	75
2.15.2.1	<a href="#">AMD_DBGAPI_ADDRESS_CLASS_NONE</a>	75
2.15.2.2	<a href="#">AMD_DBGAPI_LANE_NONE</a>	75
2.15.3	Typedef Documentation	75
2.15.3.1	<a href="#">amd_dbgapi_lane_id_t</a>	75
2.15.3.2	<a href="#">amd_dbgapi_segment_address_t</a>	75
2.15.4	Enumeration Type Documentation	75
2.15.4.1	<a href="#">amd_dbgapi_address_class_info_t</a>	76
2.15.4.2	<a href="#">amd_dbgapi_address_class_state_t</a>	76
2.15.4.3	<a href="#">amd_dbgapi_address_space_access_t</a>	76
2.15.4.4	<a href="#">amd_dbgapi_address_space_alias_t</a>	76
2.15.4.5	<a href="#">amd_dbgapi_address_space_info_t</a>	77
2.15.4.6	<a href="#">amd_dbgapi_memory_precision_t</a>	77
2.15.5	Function Documentation	77
2.15.5.1	<a href="#">amd_dbgapi_address_is_in_address_class</a>	77
2.15.5.2	<a href="#">amd_dbgapi_address_space_get_info</a>	78
2.15.5.3	<a href="#">amd_dbgapi_address_spaces_may_alias</a>	79
2.15.5.4	<a href="#">amd_dbgapi_architecture_address_class_get_info</a>	80
2.15.5.5	<a href="#">amd_dbgapi_architecture_address_class_list</a>	81
2.15.5.6	<a href="#">amd_dbgapi_architecture_address_space_list</a>	81
2.15.5.7	<a href="#">amd_dbgapi_convert_address_space</a>	82

2.15.5.8	<a href="#">amd_dbgapi_dwarf_address_class_to_address_class</a>	83
2.15.5.9	<a href="#">amd_dbgapi_dwarf_address_space_to_address_space</a>	84
2.15.5.10	<a href="#">amd_dbgapi_read_memory</a>	85
2.15.5.11	<a href="#">amd_dbgapi_set_memory_precision</a>	86
2.15.5.12	<a href="#">amd_dbgapi_write_memory</a>	87
2.16	<a href="#">Events</a>	89
2.16.1	<a href="#">Detailed Description</a>	90
2.16.2	<a href="#">Macro Definition Documentation</a>	90
2.16.2.1	<a href="#">AMD_DBGAPI_EVENT_NONE</a>	90
2.16.3	<a href="#">Enumeration Type Documentation</a>	90
2.16.3.1	<a href="#">amd_dbgapi_event_info_t</a>	90
2.16.3.2	<a href="#">amd_dbgapi_event_kind_t</a>	90
2.16.3.3	<a href="#">amd_dbgapi_runtime_state_t</a>	92
2.16.4	<a href="#">Function Documentation</a>	92
2.16.4.1	<a href="#">amd_dbgapi_event_get_info</a>	92
2.16.4.2	<a href="#">amd_dbgapi_event_processed</a>	93
2.16.4.3	<a href="#">amd_dbgapi_next_pending_event</a>	93
2.17	<a href="#">Logging</a>	95
2.17.1	<a href="#">Detailed Description</a>	95
2.17.2	<a href="#">Enumeration Type Documentation</a>	95
2.17.2.1	<a href="#">amd_dbgapi_log_level_t</a>	95
2.17.3	<a href="#">Function Documentation</a>	95
2.17.3.1	<a href="#">amd_dbgapi_set_log_level</a>	95
2.18	<a href="#">Callbacks</a>	98
2.18.1	<a href="#">Detailed Description</a>	99
2.18.2	<a href="#">Macro Definition Documentation</a>	99
2.18.2.1	<a href="#">AMD_DBGAPI_BREAKPOINT_NONE</a>	99
2.18.2.2	<a href="#">AMD_DBGAPI_SHARED_LIBRARY_NONE</a>	99
2.18.3	<a href="#">Typedef Documentation</a>	99
2.18.3.1	<a href="#">amd_dbgapi_callbacks_t</a>	99
2.18.3.2	<a href="#">amd_dbgapi_client_thread_id_t</a>	99
2.18.4	<a href="#">Enumeration Type Documentation</a>	99
2.18.4.1	<a href="#">amd_dbgapi_breakpoint_action_t</a>	99
2.18.4.2	<a href="#">amd_dbgapi_breakpoint_state_t</a>	100
2.18.4.3	<a href="#">amd_dbgapi_shared_library_state_t</a>	100
2.18.5	<a href="#">Function Documentation</a>	100
2.18.5.1	<a href="#">amd_dbgapi_report_breakpoint_hit</a>	100



2.18.5.2	amd_dbgapi_report_shared_library	101
<b>3</b>	<b>Data Structure Documentation</b>	<b>103</b>
3.1	amd_dbgapi_address_class_id_t Struct Reference	103
3.1.1	Detailed Description	103
3.1.2	Field Documentation	103
3.1.2.1	handle	103
3.2	amd_dbgapi_address_space_id_t Struct Reference	103
3.2.1	Detailed Description	104
3.2.2	Field Documentation	104
3.2.2.1	handle	104
3.3	amd_dbgapi_agent_id_t Struct Reference	104
3.3.1	Detailed Description	104
3.3.2	Field Documentation	104
3.3.2.1	handle	104
3.4	amd_dbgapi_architecture_id_t Struct Reference	105
3.4.1	Detailed Description	105
3.4.2	Field Documentation	105
3.4.2.1	handle	105
3.5	amd_dbgapi_breakpoint_id_t Struct Reference	105
3.5.1	Detailed Description	105
3.5.2	Field Documentation	105
3.5.2.1	handle	105
3.6	amd_dbgapi_callbacks_s Struct Reference	106
3.6.1	Detailed Description	106
3.6.2	Field Documentation	107
3.6.2.1	add_breakpoint	107
3.6.2.2	allocate_memory	107
3.6.2.3	deallocate_memory	107
3.6.2.4	disable_notify_shared_library	108
3.6.2.5	enable_notify_shared_library	108
3.6.2.6	get_os_pid	108
3.6.2.7	get_symbol_address	109
3.6.2.8	log_message	109
3.6.2.9	remove_breakpoint	109
3.6.2.10	set_breakpoint_state	110
3.7	amd_dbgapi_code_object_id_t Struct Reference	110

3.7.1	Detailed Description	111
3.7.2	Field Documentation	111
3.7.2.1	handle	111
3.8	amd_dbgapi_dispatch_id_t Struct Reference	111
3.8.1	Detailed Description	111
3.8.2	Field Documentation	111
3.8.2.1	handle	111
3.9	amd_dbgapi_displaced_stepping_id_t Struct Reference	111
3.9.1	Detailed Description	112
3.9.2	Field Documentation	112
3.9.2.1	handle	112
3.10	amd_dbgapi_event_id_t Struct Reference	112
3.10.1	Detailed Description	112
3.10.2	Field Documentation	112
3.10.2.1	handle	112
3.11	amd_dbgapi_process_id_t Struct Reference	112
3.11.1	Detailed Description	113
3.11.2	Field Documentation	113
3.11.2.1	handle	113
3.12	amd_dbgapi_queue_id_t Struct Reference	113
3.12.1	Detailed Description	113
3.12.2	Field Documentation	113
3.12.2.1	handle	113
3.13	amd_dbgapi_register_class_id_t Struct Reference	113
3.13.1	Detailed Description	114
3.13.2	Field Documentation	114
3.13.2.1	handle	114
3.14	amd_dbgapi_register_id_t Struct Reference	114
3.14.1	Detailed Description	114
3.14.2	Field Documentation	114
3.14.2.1	handle	114
3.15	amd_dbgapi_shared_library_id_t Struct Reference	115
3.15.1	Detailed Description	115
3.15.2	Field Documentation	115
3.15.2.1	handle	115
3.16	amd_dbgapi_wave_id_t Struct Reference	115
3.16.1	Detailed Description	115

---

3.16.2	Field Documentation	115
3.16.2.1	handle	115
<b>4</b>	<b>File Documentation</b>	<b>117</b>
4.1	include/amd-dbgapi.h File Reference	117
4.1.1	Detailed Description	128
4.1.2	Macro Definition Documentation	128
4.1.2.1	AMD_DBGAPI	128
4.1.2.2	AMD_DBGAPI_CALL	128
4.1.2.3	AMD_DBGAPI_EXPORT	128
4.1.2.4	AMD_DBGAPI_IMPORT	128
<b>Index</b>		<b>129</b>



# Chapter 1

## Introduction

The `amd-dbgapi` is a library that implements an AMD GPU debugger application programming interface (API). It provides the support necessary for a client of the library to control the execution and inspect the state of supported commercially available AMD GPU devices.

The term *client* is used to refer to the application that uses this API.

The term *library* is used to refer to the implementation of this interface being used by the client.

The term *AMD GPU* is used to refer to commercially available AMD GPU devices supported by the library.

The term *inferior* is used to refer to the process being debugged.

The library does not provide any operations to perform symbolic mappings, code object decoding, or stack unwinding. The client must use the AMD GPU code object ELF ABI defined in [User Guide for AMDGPU Backend – Code Object](#), together with the AMD GPU debug information DWARF and call frame information CFI ABI define in [User Guide for AMDGPU Backend – Code Object – DWARF](#) to perform those tasks.

The library does not provide operations for inserting or managing breakpoints. The client must write the architecture specific breakpoint instruction provided by the [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_BREAKPOINT\\_INSTRUCTION](#) query into the loaded code object memory to set breakpoints. For resuming from breakpoints the client must use the displaced stepping mechanism provided by [amd\\_dbgapi\\_displaced\\_stepping\\_start](#) and [amd\\_dbgapi\\_displaced\\_stepping\\_complete](#) in conjunction with the [amd\\_dbgapi\\_wave\\_resume](#) in single step mode. In order to determine the location of stopped waves the client must read the architecture specific program counter register available using the [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_PC\\_REGISTER](#) query and adjust it by the amount specified by the [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_BREAKPOINT\\_INSTRUCTION\\_PC\\_ADJUST](#) query.

Note that there is no way to prevent new waves being created, or to stop new waves before they start execution. So breakpoint processing should not rely on stopping all threads. Instead, the breakpoint instruction should be left inserted, and waves should be resumed using displaced stepping buffers. This will prevent breakpoints from being missed by newly created waves while resuming other waves. See [Displaced Stepping](#).

The client is responsible for checking that only a single thread at a time invokes a function provided by the library. A callback (see [Callbacks](#)) invoked by the library must not itself invoke any function provided by the library.

The library implementation creates an internal native operating system thread for its own internal use.

The library uses opaque handles to refer to the entities that it manages. These should not be modified directly. See the handle definitions for information on the lifetime and scope of handles of that type. If a handle becomes invalidated it is undefined to use it with any library operations. A handle value is unique within its scope for the lifetime of its owning entity. This is true even if the handle becomes invalidated: handle values are not reused within their scope and lifetime. Every handle with `handle` of 0 is reserved to indicate the handle does not reference an entity.

For example, a wave handle type is unique within a process. Every wave handle relating to a process will have a unique value but may have the same value as wave handles of another process. No wave handle will have the same value of

another wave handle for the same process, even if the wave handle is invalidated due to the wave terminating. When the process is detached its lifetime ends and all associated wave handles lifetime ends.

When the library is first loaded it is in the uninitialized state with the logging level set to `AMD_DBGAPI_LOG_LEVEL_NONE`.

## AMD GPU Execution Model

In this section the AMD GPU execution model is described to provide background to the reader if they are not familiar with this environment. The AMD GPU execution model is more complicated than that of a traditional CPU because of how GPU hardware is used to accelerate and schedule the very large number of threads of execution that are created on GPUs.

Chapter 2 of the [HSA Programmer's Reference Manual][hsa-prm] provides an introduction to this execution model. Note that the ROCm compilers compile directly to ISA and do not use the HSAIL intermediate language. However, the ROCr low-level runtime and ROCgdb debugger use the same terminology.

In this model, a CPU process may interact with multiple AMD GPU devices, which are termed agents. A PASSID is created for each process that interacts with agents. An agent can be executing code for multiple processes at once. This is achieved by mapping the PASSID to one of a limited set of VMIDs. Each VMID is associated with its own page table.

The AMD GPU device driver for Linux, termed the Kernel Mode Driver (KMD), manages the page tables used by each GPU so they correlate with the CPU page table for the corresponding process. The CPU and GPU page tables do not necessarily map all the same memory pages but pages they do have in common have the same virtual address. Therefore, the CPU and GPUs have a unified address space.

Each GPU includes one or more Microcode Engines (ME) that can execute microcode firmware. This firmware includes a Hardware Scheduler (HWS) that, in collaboration with the KMD, manages which processes, identified by PASSID, are mapped onto the GPU using one of the limited VMIDs. This mapping configures the VMID to use the GPU page table that corresponds to the PASSID. In this way, the code executing on the GPU from different processes is isolated.

Multiple software submission queues may be created for each agent. The GPU hardware has a limited number of pipes, each of which has a fixed number of hardware queues. The HWS, in collaboration with the KMD, is responsible for mapping software queues onto hardware queues. This is done by multiplexing the software queues onto hardware queues using time slicing. The software queues provide a virtualized abstraction, allowing for more queues than are directly supported by the hardware. Each ME manages its own set of pipes and their associated hardware queues.

To execute code on the GPU, a packet must be created and placed in a software queue. This is achieved using regular user space atomic memory operations. No Linux kernel call is required. For this reason, the queues are termed user mode queues.

ROCm uses the Asynchronous Queuing Language (AQL) packet format defined in the [HSA Platform System Architecture Specification][hsa-sysarch]. Packets can request GPU management actions (for example, manage memory coherence) and the execution of kernel functions. The ME firmware includes the Command Processor (CP) which, together with fixed-function hardware support, is responsible for detecting when packets are added to software queues that are mapped to hardware queues. Once detected, CP is responsible for initiating actions requested by the packet, using the appropriate VMID when performing all memory operations.

Dispatch packets are used to request the execution of a kernel function. Each dispatch packet specifies the address of a kernel descriptor, the address of the kernel argument block holding the arguments to the kernel function, and the number of threads of execution to create to execute the kernel function. The kernel descriptor describes how the CP must configure the hardware to execute the kernel function and the starting address of the kernel function code. The compiler generates a kernel descriptor in the code object for each kernel function and determines the kernel argument block layout. The number of threads of execution is specified as a grid, such that each thread of execution can identify its position in the grid. Conceptually, each of these threads executes the same kernel code, with the same arguments.

The dispatch grid is organized as a three-dimensional collection of work-groups, where each work-group is the same

size (except for potential boundary partial work-groups). The work-groups form a three-dimensional collection of work-items. The work-items are the threads of execution. The position of a work-item is its zero-based three-dimensional position in a work-group, termed its work-item ID, plus its work-group's three-dimensional position in the dispatch grid, termed its work-group ID. These three-dimensional IDs can also be expressed as a zero-based one-dimensional ID, termed a flat ID, by simply numbering the elements in a natural manner akin to linearizing a multi-dimensional array.

Consecutive work-items, in flat work-item ID order, of a work-group are organized into fixed size wavefronts, or waves for short. Each work-item position in the wave is termed a lane, and has a zero-base lane ID. The hardware imposes an upper limit on the number of work-items in a work-group but does not limit the number of work-groups in a dispatch grid. The hardware executes instructions for waves independently. But the lanes of a wave all execute the same instruction jointly. This is termed Single Instruction Multiple Thread (SIMT) execution.

Each hardware wave has a set of registers that are shared by all lanes of the wave, termed scalar registers. There is only one set of scalar registers for the whole wave. Instructions that act on the whole wave, which typically use scalar registers, are termed scalar instructions.

Additionally, each wave also has a set of vector registers that are replicated so each lane has its own copy. A set of vector registers can be viewed as a vector with each element of the vector belonging to the corresponding lane of the wave. Instructions that act on vector registers, which produce independent results for each lane, are termed vector instructions.

Each hardware wave has an execution mask that controls if the execution of a vector instruction should change the state of a particular lane. If the lane is masked off, no changes are made for that lane and the instruction is effectively ignored. The compiler generates code to update the execution mask which emulates independent work-item execution. However, the lanes of a wave do not execute instructions independently. If two subsets of lanes in a wave need to execute different code, the compiler will generate code to set the execution mask to execute the subset of lanes for one path, then generate instructions for that path. The compiler will then generate code to change the execution mask to enable the other subset of lanes, then generate code for those lanes. If both subsets of lanes execute the same code, the compiler will generate code to set the execution mask to include both subsets of lanes, then generate code as usual. When only a subset of lanes is enabled, they are said to be executing divergent control flow. When all lanes are enabled, they are said to be executing wave uniform control flow.

Not all MEs have the hardware to execute kernel functions. One such ME is used to execute the HWS microcode and to execute microcode that manages a service queue that is used to update GPU state. If the ME does support kernel function execution it uses fixed-function hardware to initiate the creation of waves. This is accomplished by sending requests to create work-groups to one or more Compute Units (CUs). Requests are sent to create all the work-groups of a dispatch grid. Each CU has resources to hold a fixed number of waves and has fixed-function hardware to schedule execution of these waves. The scheduler may execute multiple waves concurrently and will hide latency by switching between the waves that are ready to execute. At any point of time, a subset of the waves belonging to work-groups in a dispatch may be actively executing. As waves complete, the waves of subsequent work-group requests are created.

Each CU has a fixed amount of memory from which it allocates vector and scalar registers. The kernel descriptor specifies how many registers to allocate for a wave. There is a tradeoff between how many waves can be created on a CU and the number of registers each can use.

The CU also has a fixed size Local Data Store (LDS). A dispatch packet specifies how much LDS each work-group is allocated. All waves in a work-group are created on the same CU. This allows the LDS to be used to share data between the waves of the same work-group. There is a tradeoff between how much LDS a work-group can allocate, and the number of work-groups that can fit on a CU. The address of a location in a work-group LDS allocation is zero-based and is a different address space than the global virtual memory. There are specific instructions that take an LDS address to access it. There are also flat address instructions that map the LDS address range into an unused fixed aperture range of the global virtual address range. An LDS address can be converted to or from a flat address by offsetting by the base of the aperture. Note that a flat address in the LDS aperture only accesses the LDS work-group allocation for the wave that uses it. The same address will access different LDS allocations if used by waves in different work-groups.

The dispatch packet specifies the amount of scratch memory that must be allocated for a work-item. This is used for work-item private memory. Fixed-function hardware in the CU manages per wave allocation of scratch memory from pre-allocated global virtual memory mapped to GPU device memory. Like an LDS address, a scratch address is zero-

based, but is per work-item instead of per work-group. It maps to an aperture in a flat address. The hardware swizzles this address so that adjacent lanes access adjacent DWORDs (4 bytes) in global memory for better cache performance.

For an AMD Vega 10 GPU the work-group size limit is 1,024 work-items, the wavefront size is 64, and the CU count is 64. A CU can hold up to 40 waves (this is limited to 32 if using scratch memory). Therefore, a work-group can comprise between 1 and 16 waves inclusive, and there can be up to 2,560 waves, making a maximum of 163,840 work-items. A CU is organized as 4 SIMDs that can each hold 10 waves. Each SIMD has 256 DWORD vector registers and 800 scalar registers. A single wave can access up to 256 vector registers and 112 scalar registers. A CU has 64KiB of LDS.

## References

1. Advanced Micro Devices: [www.amd.com](http://www.amd.com)
2. AMD ROCm Platform: [rocm-documentation.readthedocs.io/en/latest](http://rocm-documentation.readthedocs.io/en/latest)
3. Bus:Device.Function (BDF) Notation: [wiki.xen.org/wiki/Bus:Device.Function\\_\(BDF\)\\_-\\_Notation\\_Notation](http://wiki.xen.org/wiki/Bus:Device.Function_(BDF)_-_Notation_Notation)
4. HSA Platform System Architecture Specification: [www.hsafoundation.com/html\\_spec111/HSA\\_Library.htm::SysArch/Topics/SysArch\\_title\\_page.htm](http://www.hsafoundation.com/html_spec111/HSA_Library.htm::SysArch/Topics/SysArch_title_page.htm)
5. HSA Programmer's Reference Manual: [www.hsafoundation.com/html\\_spec111/HSA\\_Library.htm::PRM/Topics/PRM\\_title\\_page.htm](http://www.hsafoundation.com/html_spec111/HSA_Library.htm::PRM/Topics/PRM_title_page.htm)
6. Semantic Versioning: [semver.org](http://semver.org)
7. The LLVM Compiler Infrastructure: [llvm.org](http://llvm.org)
8. User Guide for AMDGPU LLVM Backend: [llvm.org/docs/AMDGPUUsage.html](http://llvm.org/docs/AMDGPUUsage.html)



## Chapter 2

# Module Documentation

### 2.1 Basic Types

Types used for common properties.

#### Typedefs

- typedef uint64\_t [amd\\_dbgapi\\_global\\_address\\_t](#)  
*Integral type used for a global virtual memory address in the inferior process.*
- typedef uint64\_t [amd\\_dbgapi\\_size\\_t](#)  
*Integral type used for sizes, including memory allocations, in the inferior.*
- typedef pid\_t [amd\\_dbgapi\\_os\\_pid](#)  
*Native operating system process id.*
- typedef int [amd\\_dbgapi\\_notifier\\_t](#)  
*Type used to notify the client of the library that a process may have pending events.*

#### Enumerations

- enum [amd\\_dbgapi\\_changed\\_t](#) { [AMD\\_DBGAPI\\_CHANGED\\_NO](#) = 0, [AMD\\_DBGAPI\\_CHANGED\\_YES](#) = 1 }  
*Indication of if a value has changed.*

#### 2.1.1 Detailed Description

Types used for common properties.

#### 2.1.2 Typedef Documentation

##### 2.1.2.1 typedef uint64\_t [amd\\_dbgapi\\_global\\_address\\_t](#)

Integral type used for a global virtual memory address in the inferior process.

### 2.1.2.2 typedef int amd\_dbgapi\_notifier\_t

Type used to notify the client of the library that a process may have pending events.

A notifier is created when [amd\\_dbgapi\\_process\\_attach](#) is used to successfully attach to a process. It is obtained using the [AMD\\_DBGAPI\\_PROCESS\\_INFO\\_NOTIFIER](#) query. If the notifier indicates there may be pending events, then [amd\\_dbgapi\\_next\\_pending\\_event](#) can be used to retrieve them.

For Linux<sup>®</sup> this is a file descriptor number that can be used with the `poll` call to wait on events from multiple sources. The file descriptor is made to have data available when events may be added to the pending events. The client can flush the file descriptor and read the pending events until none are available. Note that the file descriptor may become ready spuriously when no pending events are available, in which case the client should simply wait again. If new pending events are added while reading the pending events, then the file descriptor will again have data available. The amount of data on the file descriptor is not an indication of the number of pending events as the file may become full and so no further data will be added. The file descriptor is simply a robust way to determine if there may be some pending events.

### 2.1.2.3 typedef pid\_t amd\_dbgapi\_os\_pid

Native operating system process id.

This is the process id used by the operating system that is executing the library. It is used in the implementation of the library to interact with the AMD GPU device driver.

### 2.1.2.4 typedef uint64\_t amd\_dbgapi\_size\_t

Integral type used for sizes, including memory allocations, in the inferior.

## 2.1.3 Enumeration Type Documentation

### 2.1.3.1 enum amd\_dbgapi\_changed\_t

Indication of if a value has changed.

Enumerator

**AMD\_DBGAPI\_CHANGED\_NO** The value has not changed.

**AMD\_DBGAPI\_CHANGED\_YES** The value has changed.

## 2.2 Status Codes

Most operations return a status code to indicate success or error.

### Enumerations

```
enum amd_dbgapi_status_t {
    AMD_DBGAPI_STATUS_SUCCESS = 0, AMD_DBGAPI_STATUS_ERROR = -1, AMD_DBGAPI_STATUS_FATAL = -2,
    AMD_DBGAPI_STATUS_ERROR_NOT_SUPPORTED = -3,
    AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT = -4, AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE = -5,
    AMD_DBGAPI_STATUS_ERROR_ALREADY_INITIALIZED = -6, AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED = -7,
    AMD_DBGAPI_STATUS_ERROR_VERSION_MISMATCH = -8, AMD_DBGAPI_STATUS_ERROR_ALREADY_ATTACHED = -9,
    AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID = -10, AMD_DBGAPI_STATUS_ERROR_ILLEGAL_INSTRUCTION = -11,
    AMD_DBGAPI_STATUS_ERROR_INVALID_CODE_OBJECT_ID = -12, AMD_DBGAPI_STATUS_ERROR_INVALID_ELF_AMDGPU_MACHINE = -13,
    AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID = -14, AMD_DBGAPI_STATUS_ERROR_INVALID_AGENT_ID = -15,
    AMD_DBGAPI_STATUS_ERROR_INVALID_QUEUE_ID = -16, AMD_DBGAPI_STATUS_ERROR_INVALID_DISPATCH_ID = -17,
    AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID = -18, AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED = -19,
    AMD_DBGAPI_STATUS_ERROR_WAVE_STOPPED = -20, AMD_DBGAPI_STATUS_ERROR_WAVE_OUTSTANDING_STOP = -21,
    AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_RESUMABLE = -22, AMD_DBGAPI_STATUS_ERROR_INVALID_DISPLACED_STEPPING_ID = -23,
    AMD_DBGAPI_STATUS_ERROR_DISPLACED_STEPPING_BUFFER_UNAVAILABLE = -24, AMD_DBGAPI_STATUS_ERROR_INVALID_WATCHPOINT_ID = -25,
    AMD_DBGAPI_STATUS_ERROR_NO_WATCHPOINT_AVAILABLE = -26, AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_CLASS_ID = -27,
    AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID = -28, AMD_DBGAPI_STATUS_ERROR_INVALID_LANE_ID = -29,
    AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_CLASS_ID = -30, AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID = -31,
    AMD_DBGAPI_STATUS_ERROR_MEMORY_ACCESS = -32, AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_CONVERSION = -33,
    AMD_DBGAPI_STATUS_ERROR_INVALID_EVENT_ID = -34, AMD_DBGAPI_STATUS_ERROR_INVALID_SHARED_LIBRARY_ID = -35,
    AMD_DBGAPI_STATUS_ERROR_INVALID_BREAKPOINT_ID = -36, AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK = -37,
    AMD_DBGAPI_STATUS_ERROR_INVALID_CLIENT_PROCESS_ID = -38, AMD_DBGAPI_STATUS_ERROR_PROCESS_EXITED = -39,
    AMD_DBGAPI_STATUS_ERROR_LIBRARY_NOT_LOADED = -40, AMD_DBGAPI_STATUS_ERROR_SYMBOL_NOT_FOUND = -41,
    AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS = -42 };
```

*AMD debugger API status codes.*

### Functions

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_get_status_string (amd_dbgapi_status_t status, const char **status_string)`

*Query a textual description of a status code.*

#### 2.2.1 Detailed Description

Most operations return a status code to indicate success or error.

## 2.2.2 Enumeration Type Documentation

### 2.2.2.1 enum amd\_dbgapi\_status\_t

AMD debugger API status codes.

Enumerator

**AMD\_DBGAPI\_STATUS\_SUCCESS** The function has executed successfully.

**AMD\_DBGAPI\_STATUS\_ERROR** A generic error has occurred.

**AMD\_DBGAPI\_STATUS\_FATAL** A fatal error has occurred. The library encountered an error from which it cannot recover. All processes are detached. All breakpoints added by [amd\\_dbgapi\\_callbacks\\_s::add\\_breakpoint](#) are attempted to be removed. All handles are invalidated. The library is left in an uninitialized state. The logging level is reset to [AMD\\_DBGAPI\\_LOG\\_LEVEL\\_NONE](#).

To resume using the library the client must re-initialize the library; re-attach to any processes; re-fetch the list of code objects, agents, queues, dispatches, and waves; and update the state of all waves as appropriate. While in the uninitialized state the inferior processes will continue executing but any execution of a breakpoint instruction will put the queue into an error state, aborting any executing waves. Note that recovering from a fatal error most likely will require the user of the client to re-start their session.

The cause of possible fatal errors is that resources became exhausted or unique handle numbers became exhausted.

**AMD\_DBGAPI\_STATUS\_ERROR\_NOT\_SUPPORTED** The operation is not supported.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARGUMENT** An invalid argument was given to the function.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARGUMENT\_SIZE** An invalid size was given to the function.

**AMD\_DBGAPI\_STATUS\_ERROR\_ALREADY\_INITIALIZED** The library is already initialized.

**AMD\_DBGAPI\_STATUS\_ERROR\_NOT\_INITIALIZED** The library is not initialized.

**AMD\_DBGAPI\_STATUS\_ERROR\_VERSION\_MISMATCH** The version of the kernel driver does not match the version required by the library.

**AMD\_DBGAPI\_STATUS\_ERROR\_ALREADY\_ATTACHED** The process is already attached to the given inferior process.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARCHITECTURE\_ID** The architecture handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_ILLEGAL\_INSTRUCTION** The bytes being disassembled are not a legal instruction.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_CODE\_OBJECT\_ID** The code object handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ELF\_AMDGPU\_MACHINE** The ELF AMD GPU machine value is invalid or unsupported.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_PROCESS\_ID** The process handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_AGENT\_ID** The agent handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_QUEUE\_ID** The queue handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_DISPATCH\_ID** The dispatch handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_WAVE\_ID** The wave handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_NOT\_STOPPED** The wave is not stopped.

**AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_STOPPED** The wave is stopped.

**AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_OUTSTANDING\_STOP** The wave has an outstanding stop request.

**AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_NOT\_RESUMABLE** The wave cannot be resumed.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_DISPLACED\_STEPPING\_ID** The displaced stepping handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_DISPLACED\_STEPPING\_BUFFER\_UNAVAILABLE** No more displaced stepping buffers are available that are suitable for the requested wave.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_WATCHPOINT\_ID** The watchpoint handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_NO\_WATCHPOINT\_AVAILABLE** No more watchpoints available.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_REGISTER\_CLASS\_ID** The register class handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_REGISTER\_ID** The register handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_LANE\_ID** The lane handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_CLASS\_ID** The address class handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_SPACE\_ID** The address space handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_MEMORY\_ACCESS** An error occurred while trying to access memory in the inferior.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_SPACE\_CONVERSION** The segment address cannot be converted to the requested address space.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_EVENT\_ID** The event handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_SHARED\_LIBRARY\_ID** The shared library handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_BREAKPOINT\_ID** The breakpoint handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_CLIENT\_CALLBACK** A callback to the client reported an error.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_CLIENT\_PROCESS\_ID** The client process handle is invalid.

**AMD\_DBGAPI\_STATUS\_ERROR\_PROCESS\_EXITED** The native operating system process associated with a client process has exited.

**AMD\_DBGAPI\_STATUS\_ERROR\_LIBRARY\_NOT\_LOADED** The shared library is not currently loaded.

**AMD\_DBGAPI\_STATUS\_ERROR\_SYMBOL\_NOT\_FOUND** The symbol was not found.

**AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS** The address is not within the shared library.

### 2.2.3 Function Documentation

**2.2.3.1** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_get_status_string ( amd_dbgapi_status_t status, const char ** status_string )`

Query a textual description of a status code.

This function can be used even when the library is uninitialized.

#### Parameters

in	<i>status</i>	Status code.
out	<i>status_string</i>	A NUL terminated string that describes the status code. The string is read only and owned by the library.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully. <i>status_string</i> has been updated.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>status</i> is an invalid status code or <i>status_string</i> is NULL. <i>status_string</i> is unaltered.

## 2.3 Versioning

Version information about the interface and the associated installed library.

### Macros

- `#define AMD_DBGAPI_SYMBOL_VERSION "_0.1"`  
*The name used for the shared library versioned symbols.*

### Enumerations

- `enum { AMD_DBGAPI_VERSION_MAJOR = 0, AMD_DBGAPI_VERSION_MINOR = 1 }`  
*The semantic version of the interface following [semver.org][semver] rules.*

### Functions

- `void AMD_DBGAPI amd_dbgapi_get_version (uint32_t *major, uint32_t *minor, uint32_t *patch)`  
*Query the version of the installed library.*
- `const char AMD_DBGAPI * amd_dbgapi_get_build_name (void)`  
*Query the installed library build name.*

#### 2.3.1 Detailed Description

Version information about the interface and the associated installed library.

#### 2.3.2 Macro Definition Documentation

##### 2.3.2.1 `#define AMD_DBGAPI_SYMBOL_VERSION "_0.1"`

The name used for the shared library versioned symbols.

After dynamically loading the shared library (using `dlopen`), this can be used to specify the version of the symbols (using `dlvsym`) that match this interface version. An error will be reported if the installed library does not support this interface version.

#### 2.3.3 Enumeration Type Documentation

##### 2.3.3.1 anonymous enum

The semantic version of the interface following [semver.org][semver] rules.

A client that uses this interface is only compatible with the installed library if the major version numbers match and the interface minor version number is less than or equal to the installed library minor version number.

### Enumerator

**AMD\_DBGAPI\_VERSION\_MAJOR** The major version of the interface.

**AMD\_DBGAPI\_VERSION\_MINOR** The minor version of the interface.

### 2.3.4 Function Documentation

#### 2.3.4.1 `const char AMD_DBGAPI* amd_dbgapi_get_build_name ( void )`

Query the installed library build name.

This function can be used even when the library is not initialized.

##### Returns

Returns a string describing the build version of the library. The string is owned by the library.

#### 2.3.4.2 `void AMD_DBGAPI amd_dbgapi_get_version ( uint32_t * major, uint32_t * minor, uint32_t * patch )`

Query the version of the installed library.

Return the version of the installed library. This can be used to check if it is compatible with this interface version. This function can be used even when the library is not initialized.

##### Parameters

out	<i>major</i>	The major version number is stored if non-NULL.
out	<i>minor</i>	The minor version number is stored if non-NULL.
out	<i>patch</i>	The patch version number is stored if non-NULL.

## 2.4 Initialization and Finalization

Operations to control initializing and finalizing the library.

### Functions

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_initialize (amd_dbgapi_callbacks_t *callbacks)`  
*Initialize the library.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_finalize (void)`  
*Finalize the library.*

### 2.4.1 Detailed Description

Operations to control initializing and finalizing the library. When the library is first loaded it is in the uninitialized state. Before any operation can be used, the library must be initialized. The exception is the status operation in [Status Codes](#) and the version operations in [Versioning](#) which can be used regardless of whether the library is initialized.

### 2.4.2 Function Documentation

#### 2.4.2.1 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_finalize ( void )`

Finalize the library.

Finalizing the library invalidates all handles previously returned by any operation. It is undefined to use any such handle even if the library is subsequently initialized with [amd\\_dbgapi\\_initialize](#). Finalizing the library implicitly detaches from any processes currently attached. It is allowed to initialize and finalize the library multiple times. Finalizing the library does not changed the logging level (see [Logging](#)).

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the library is now uninitialized.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if any of the <a href="#">amd_dbgapi_callbacks_s</a> callbacks used return an error. The library is still left uninitialized, but the client may be in an inconsistent state.

#### 2.4.2.2 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_initialize ( amd_dbgapi_callbacks_t * callbacks )`

Initialize the library.

Initialize the library so that the library functions can be used to control the AMD GPU devices accessed by processes.

Initializing the library does not change the logging level (see [Logging](#)).



## Parameters

<code>in</code>	<code>callbacks</code>	A set of callbacks must be provided. These are invoked by certain operations. They are described in <a href="#">amd_dbgapi_callbacks_t</a> .
-----------------	------------------------	--

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the library is now initialized.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library remains uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_ALREADY_INITIALIZED</a>	The library is already initialized. The library is left initialized and the callbacks are not changed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>callbacks</code> is NULL or has fields that are NULL. The library remains uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if any of the <a href="#">amd_dbgapi_callbacks_s</a> callbacks used return an error. The library remains uninitialized.

## 2.5 Architectures

Operations related to AMD GPU architectures.

### Data Structures

- struct `amd_dbgapi_architecture_id_t`

*Opaque architecture handle.*

### Macros

- #define `AMD_DBGAPI_ARCHITECTURE_NONE` (`amd_dbgapi_architecture_id_t`{ 0 })

*The NULL architecture handle.*

### Enumerations

- enum `amd_dbgapi_architecture_info_t` {  
`AMD_DBGAPI_ARCHITECTURE_INFO_NAME` = 1, `AMD_DBGAPI_ARCHITECTURE_INFO_ELF_AMDGPU_MACHINE` = 2, `AMD_DBGAPI_ARCHITECTURE_INFO_LARGEST_INSTRUCTION_SIZE` = 3, `AMD_DBGAPI_ARCHITECTURE_INFO_MINIMUM_INSTRUCTION_ALIGNMENT` = 4,  
`AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_SIZE` = 5, `AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_PC_ADJUST` = 7, `AMD_DBGAPI_ARCHITECTURE_INFO_PC_REGISTER` = 8,  
`AMD_DBGAPI_ARCHITECTURE_INFO_EXECUTION_MASK_REGISTER` = 9, `AMD_DBGAPI_ARCHITECTURE_INFO_WATCHPOINT_COUNT` = 10, `AMD_DBGAPI_ARCHITECTURE_INFO_WATCHPOINT_SHARE` = 11, `AMD_DBGAPI_ARCHITECTURE_INFO_DEFAULT_GLOBAL_ADDRESS_SPACE` = 12,  
`AMD_DBGAPI_ARCHITECTURE_INFO_PRECISE_MEMORY_SUPPORTED` = 13 }

*Architecture queries that are supported by `amd_dbgapi_architecture_get_info`.*

### Functions

- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_architecture_get_info` (`amd_dbgapi_architecture_id_t` `architecture_id`, `amd_dbgapi_architecture_info_t` `query`, `size_t` `value_size`, `void *``value`)  
*Query information about an architecture.*
- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_get_architecture` (`uint32_t` `elf_amdgpu_machine`, `amd_dbgapi_architecture_id_t *``architecture_id`)  
*Get an architecture from the AMD GPU ELF `EF_AMDGPU_MACH` value corresponding to the architecture.*
- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_disassemble_instruction` (`amd_dbgapi_architecture_id_t` `architecture_id`, `amd_dbgapi_global_address_t` `address`, `amd_dbgapi_size_t *``size`, `const void *``memory`, `char **``instruction_text`, `size_t *``address_operand_count`, `amd_dbgapi_global_address_t **``address_operands`)  
*Disassemble a single instruction.*

#### 2.5.1 Detailed Description

Operations related to AMD GPU architectures. The library supports a family of AMD GPU devices. Each device has its own architectural properties. The operations in this section provide information about the supported architectures.

## 2.5.2 Macro Definition Documentation

### 2.5.2.1 #define AMD\_DBGAPI\_ARCHITECTURE\_NONE (amd\_dbgapi\_architecture\_id\_t{ 0 })

The NULL architecture handle.

## 2.5.3 Enumeration Type Documentation

### 2.5.3.1 enum amd\_dbgapi\_architecture\_info\_t

Architecture queries that are supported by [amd\\_dbgapi\\_architecture\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_architecture\\_get\\_info](#).

#### Enumerator

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_NAME** Return the architecture name. The type of this attribute is a pointer to a NUL terminated `char*`. It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_ELF\_AMDGPU\_MACHINE** Return the AMD GPU ELF `EF_AMDGPU_MACHINE` value corresponding to the architecture. This is defined as a bit field in the `e_flags` AMD GPU ELF header. See [User Guide for AMDGPU Backend - Code Object - Header](#). The type of this attribute is `uint32_t`.

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_LARGEST\_INSTRUCTION\_SIZE** Return the largest instruction size in bytes for the architecture. The type of this attribute is [amd\\_dbgapi\\_size\\_t](#).

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_MINIMUM\_INSTRUCTION\_ALIGNMENT** Return the minimum instruction alignment in bytes for the architecture. The returned value will be a power of two. The type of this attribute is [amd\\_dbgapi\\_size\\_t](#).

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION\_SIZE** Return the breakpoint instruction size in bytes for the architecture. The type of this attribute is [amd\\_dbgapi\\_size\\_t](#).

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION** Return the breakpoint instruction for the architecture. The type of this attribute is pointer to `N` bytes where `N` is the value returned by the [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_BREAKPOINT\\_INSTRUCTION\\_SIZE](#) query. It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION\_PC\_ADJUST** Return the number of bytes to subtract from the PC after stopping due to a breakpoint instruction to get the address of the breakpoint instruction for the architecture. The type of this attribute is [amd\\_dbgapi\\_size\\_t](#).

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_PC\_REGISTER** Return the register handle for the PC for the architecture. The type of this attribute is [amd\\_dbgapi\\_register\\_id\\_t](#).

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_EXECUTION\_MASK\_REGISTER** Return the register handle for the execution mask for the architecture. The type of this attribute is [amd\\_dbgapi\\_register\\_id\\_t](#). Return [AMD\\_DBGAPI\\_REGISTER\\_NONE](#) if the architecture does not use an execution mask.

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_WATCHPOINT\_COUNT** Return the number of data watchpoints supported by the architecture. Zero is returned if data watchpoints are not supported. The type of this attribute is `size_t`.

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_WATCHPOINT\_SHARE** Return how watchpoints are shared between processes. The type of this attribute is `uint32_t` with the values defined by [amd\\_dbgapi\\_watchpoint\\_share\\_kind\\_t](#).

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_DEFAULT\_GLOBAL\_ADDRESS\_SPACE** Return the default address space for global memory. The type of this attribute is [amd\\_dbgapi\\_address\\_space\\_id\\_t](#).

**AMD\_DBGAPI\_ARCHITECTURE\_INFO\_PRECISE\_MEMORY\_SUPPORTED** Return if the architecture supports controlling memory precision. The type of this attribute is `uint32_t` with the values defined by [amd\\_dbgapi\\_memory\\_precision\\_t](#).

## 2.5.4 Function Documentation

**2.5.4.1** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_architecture_get_info ( amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_architecture_info_t query, size_t value_size, void * value )`

Query information about an architecture.

[amd\\_dbgapi\\_architecture\\_info\\_t](#) specifies the queries supported and the type returned using the `value` argument.

### Parameters

in	<i>architecture_id</i>	The architecture being queried.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <code>value</code> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <code>value</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<code>architecture_id</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>value</code> is NULL or <code>query</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<code>value_size</code> does not match the size of the result. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <code>value</code> returns NULL. <code>value</code> is unaltered.

**2.5.4.2** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_disassemble_instruction ( amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_global_address_t address, amd_dbgapi_size_t * size, const void * memory, char ** instruction_text, size_t * address_operand_count, amd_dbgapi_global_address_t ** address_operands )`

Disassemble a single instruction.

### Parameters

in	<i>architecture_id</i>	The architecture to use to perform the disassembly.
----	------------------------	---

in	<i>address</i>	The address of the first byte of the instruction.
in,out	<i>size</i>	Pass in the number of bytes available in <code>memory</code> which must be greater than 0. Return the number of bytes consumed to decode the instruction.
in	<i>memory</i>	The bytes to decode as an instruction. Must point to an array of at least <code>size</code> bytes. The <a href="#">AMD_DBGAPI_ARCHITECTURE_INFO_LARGEST_INSTRUCTION_SIZE</a> query for <code>architecture_id</code> can be used to determine the number of bytes of the largest instruction. By making <code>size</code> at least this size ensures that the instruction can be decoded if legal. However, <code>size</code> may need to be smaller if no memory exists at the address of <code>address</code> plus <code>size</code> .
out	<i>instruction_text</i>	Pointer to NUL terminated string that contains the disassembled textual representation of the instruction. The memory is allocated using the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.
out	<i>address_operand_count</i>	The number of operands in the disassembled instruction that were memory addresses. If NULL, no value is returned.
out	<i>address_operands</i>	Pointer to an array of <code>address_operand_count</code> <a href="#">amd_dbgapi_global_address_t</a> values. Each one corresponds to an operand of the disassembled instruction that is a memory address, ordered left to right. The client can choose to determine a symbol for the address and append it as a comment to the disassembled instruction text. The memory is allocated using the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client. If NULL, no value is returned. Either both <code>address_operand_count</code> and <code>address_operands</code> must be NULL or both must be non-NULL.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <code>size</code> , <code>instruction_text</code> , <code>address_operand_count</code> , and <code>address_operands</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>size</code> , <code>instruction_text</code> , <code>address_operand_count</code> , and <code>address_operands</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>size</code> , <code>instruction_text</code> , <code>address_operand_count</code> , and <code>address_operands</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<code>architecture_id</code> is invalid. <code>size</code> , <code>instruction_text</code> , <code>address_operand_count</code> , and <code>address_operands</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>size</code> , <code>memory</code> , or <code>instruction_text</code> are NULL; <code>size</code> is 0; or <code>address_operand_count</code> and <code>address_operands</code> are not both NULL or not both non-NULL. <code>size</code> , <code>instruction_text</code> , <code>address_operand_count</code> , and <code>address_operands</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR</a>	Encountered an error disassembling the instruction. The bytes may or may not be a legal instruction. <code>size</code> , <code>instruction_text</code> , <code>address_operand_count</code> , and <code>address_operands</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_ILLEGAL_INSTRUCTION</a>	The bytes starting at <code>address</code> , when up to <code>size</code> bytes are available, are not a legal instruction for the architecture. <code>size</code> , <code>instruction_text</code> , <code>address_operand_count</code> , and <code>address_operands</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <code>instruction_text</code> and <code>address_operands</code> returns NULL. <code>size</code> , <code>instruction_text</code> , <code>address_operand_count</code> , and <code>address_operands</code> are unaltered.

#### 2.5.4.3 `amd_dbgapi_status_t` AMD\_DBGAPI `amd_dbgapi_get_architecture` ( `uint32_t` *elf\_amdgpu\_machine*, `amd_dbgapi_architecture_id_t` \* *architecture\_id* )

Get an architecture from the AMD GPU ELF `EF_AMDGPU_MACH` value corresponding to the architecture.

This is defined as a bit field in the `e_flags` AMD GPU ELF header. See [User Guide for AMDGPU Backend - Code Object

- Header] (<https://llvm.org/docs/AMDGPUUsage.html#header>).

##### Parameters

in	<i>elf_amdgpu_machine</i>	The AMD GPU ELF <code>EF_AMDGPU_MACH</code> value.
out	<i>architecture_id</i>	The corresponding architecture.

##### Return values

<a href="#"><code>AMD_DBGAPI_STATUS_SUCCESS</code></a>	The function has been executed successfully and the result is stored in <code>architecture_id</code> .
<a href="#"><code>AMD_DBGAPI_STATUS_FATAL</code></a>	A fatal error occurred. The library is left uninitialized and <code>architecture_id</code> is unaltered.
<a href="#"><code>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</code></a>	The library is not initialized. The library is left uninitialized and <code>architecture_id</code> is unaltered.
<a href="#"><code>AMD_DBGAPI_STATUS_ERROR_INVALID_ELF_AMDGPU_MACHINE</code></a>	<code>elf_amdgpu_machine</code> is invalid or unsupported. <code>architecture_id</code> is unaltered.
<a href="#"><code>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</code></a>	<code>architecture_id</code> is NULL. <code>architecture_id</code> is unaltered.

## 2.6 Processes

Operations related to establishing AMD GPU debug control of a process.

### Data Structures

- struct `amd_dbgapi_process_id_t`  
*Opaque process handle.*

### Macros

- #define `AMD_DBGAPI_PROCESS_NONE` (`amd_dbgapi_process_id_t{ 0 }`)  
*The NULL process handle.*

### Typedefs

- typedef void \* `amd_dbgapi_client_process_id_t`  
*Opaque client process handle.*

### Enumerations

- enum `amd_dbgapi_process_info_t` { `AMD_DBGAPI_PROCESS_INFO_NOTIFIER` = 1 }  
*Process queries that are supported by `amd_dbgapi_process_get_info`.*
- enum `amd_dbgapi_progress_t` { `AMD_DBGAPI_PROGRESS_NORMAL` = 0, `AMD_DBGAPI_PROGRESS_NO_FORWARD` = 1 }  
*The kinds of progress supported by the library.*

### Functions

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_get_info` (`amd_dbgapi_process_id_t` process\_id, `amd_dbgapi_process_info_t` query, `size_t` value\_size, void \*value)  
*Query information about a process.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_attach` (`amd_dbgapi_client_process_id_t` client\_process\_id, `amd_dbgapi_process_id_t` \*process\_id)  
*Attach to a process in order to provide debug control of the AMD GPUs it uses.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_detach` (`amd_dbgapi_process_id_t` process\_id)  
*Detach from a process.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_set_progress` (`amd_dbgapi_process_id_t` process\_id, `amd_dbgapi_progress_t` progress)  
*Set the progress required for a process.*

#### 2.6.1 Detailed Description

Operations related to establishing AMD GPU debug control of a process. The library supports AMD GPU debug control of multiple operating system processes. Each process can have access to multiple AMD GPU devices, but each process uses the AMD GPU devices independently of other processes.

## 2.6.2 Macro Definition Documentation

### 2.6.2.1 `#define AMD_DBGAPI_PROCESS_NONE (amd_dbgapi_process_id_t{ 0 })`

The NULL process handle.

## 2.6.3 Typedef Documentation

### 2.6.3.1 `typedef void* amd_dbgapi_client_process_id_t`

Opaque client process handle.

A pointer to client data associated with a process. This pointer is passed to the process specific callbacks (see [Callbacks](#)) to allow the client of the library to identify the process. Each process must have a single unique value.

## 2.6.4 Enumeration Type Documentation

### 2.6.4.1 `enum amd_dbgapi_process_info_t`

Process queries that are supported by [amd\\_dbgapi\\_process\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_process\\_get\\_info](#).

Enumerator

**`AMD_DBGAPI_PROCESS_INFO_NOTIFIER`** The notifier for the process that indicates if pending events are available. The type of this attributes is [amd\\_dbgapi\\_notifier\\_t](#).

### 2.6.4.2 `enum amd_dbgapi_progress_t`

The kinds of progress supported by the library.

In performing operations, the library may make both waves it needs to access, as well as other waves, unavailable for hardware execution. After completing the operation, it will make all waves available for hardware execution. This is termed pausing and unpausing wave execution respectively. Pausing and unpausing waves for each command separately works but can result in longer latency than if several commands could be performed while the waves are paused. When debugging the very large number of waves that can exist on an AMD GPU can involve many operations, making batching commands even more beneficial. The progress setting allows controlling this behavior.

Enumerator

**`AMD_DBGAPI_PROGRESS_NORMAL`** Normal progress is needed. Commands are issued immediately. After completing each command all non-stopped waves will be unpaused. Switching from another progress mode to this will unpause any waves that are paused.

**`AMD_DBGAPI_PROGRESS_NO_FORWARD`** No forward progress is needed. Commands are issued immediately. After completing each command, non-stopped waves may be left paused. The waves left paused may include both the wave(s) the command operates on, as well as other waves. While in [AMD\\_DBGAPI\\_PROGRESS\\_NO\\_FORWARD](#) mode, paused waves may remain paused, or may be unpaused at any point. Only by leaving [AMD\\_DBGAPI\\_PROGRESS\\_NO\\_FORWARD](#) mode will the library not leave any waves paused after completing a command.

This can result in a series of commands completing far faster than in [AMD\\_DBGAPI\\_PROGRESS\\_NORMAL](#) mode. Also, any queries for lists such as [amd\\_dbgapi\\_wave\\_list](#) may return `unchanged` as true more often, reducing the work needed to parse the lists to determine what has changed. With large lists this can be



significant. If the client needs a wave to complete a single step resume, then it must leave [AMD\\_DBGAPI\\_PROGRESS\\_NO\\_FORWARD](#) mode in order to prevent that wave from remaining paused.

### 2.6.5 Function Documentation

**2.6.5.1** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_attach ( amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_process_id_t * process_id )`

Attach to a process in order to provide debug control of the AMD GPUs it uses.

Attaching can be performed on processes that have not started executing, as well as those that are already executing.

The process progress is initialized to [AMD\\_DBGAPI\\_PROGRESS\\_NORMAL](#). All agents accessed by the process are configured to [AMD\\_DBGAPI\\_MEMORY\\_PRECISION\\_NONE](#).

The client process handle must have been associated with a native operating system process, and the [amd\\_dbgapi\\_callbacks\\_s::get\\_os\\_pid](#) callback is used to obtain it.

If the associated native operating system process exits while the library is attached to it, appropriate actions are taken to reflect that the inferior process no longer has any state. For example, pending events are created for wave command termination if there are pending wave stop or wave single step requests; a pending code object list updated event is created if there were codes objects previously loaded; a pending runtime event is created to indicate the runtime support has been unloaded if previously loaded; and queries on agents, queues, dispatches, waves, and code objects will report none exist. The process handle remains valid until [amd\\_dbgapi\\_process\\_detach](#) is used to detach from the client process.

If the associated native operating system process has already exited when attaching, then the attach is still successful, but any queries on agents, queues, dispatches, waves, and code objects will report none exist.

If the associated native operating system process exits while a library operation is being executed, then the operation behaves as if the process exited before it was invoked. For example, a wave operation will report an invalid wave handle, a list query will report an empty list, and so forth.

#### Parameters

in	<i>client_process_id</i>	The client handle for the process. It is passed as an argument to any callbacks performed to indicate the process being requested.
out	<i>process_id</i>	The process handle to use for all operations related to this process.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the process is now attached returning <i>process_id</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>process_id</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>process_id</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_ALREADY_ATTACHED</a>	The process is already attached. The process remains attached and <i>process_id</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_VERSION_MISMATCH</a>	The installed AMD GPU driver version is not compatible with the library. The process is not attached and <i>process_id</i> is unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>client_process_id</code> or <code>process_id</code> are NULL. The process is not attached and <code>process_id</code> is unaltered.
--	---

#### 2.6.5.2 `amd_dbgapi_status_t` AMD\_DBGAPI `amd_dbgapi_process_detach ( amd_dbgapi_process_id_t process_id )`

Detach from a process.

The AMD GPU devices used by the process can no longer be controlled. Any waves with a displaced stepping buffer are stopped and the displaced stepping buffer completed. Any waves in the stopped or single step state are resumed in non-single step mode. Any pending events are discarded. Any data watchpoints are removed. All agents are configured to [AMD\\_DBGAPI\\_MEMORY\\_PRECISION\\_NONE](#). Execution of the inferior process will continue unaffected by the library.

It is undefined to use the process handle, or any handles returned by previous operations performed for that process, after detaching. A process can be attached and detached multiple times.

The client is responsible for removing any inserted breakpoints before detaching. Failing to do so will cause the inferior execution of a breakpoint instruction to put the queue into an error state, aborting any executing waves for dispatches on that queue.

##### Parameters

<code>process_id</code>	The process handle that is being detached.
-------------------------	--

##### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the process has been detached.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	The <code>process_id</code> is invalid. No process is detached.

#### 2.6.5.3 `amd_dbgapi_status_t` AMD\_DBGAPI `amd_dbgapi_process_get_info ( amd_dbgapi_process_id_t process_id, amd_dbgapi_process_info_t query, size_t value_size, void * value )`

Query information about a process.

[amd\\_dbgapi\\_process\\_info\\_t](#) specifies the queries supported and the type returned using the `value` argument.

##### Parameters

in	<code>process_id</code>	The process being queried.
in	<code>query</code>	The query being requested.
in	<code>value_size</code>	Size of the memory pointed to by <code>value</code> . Must be equal to the byte size of the query result.
out	<code>value</code>	Pointer to memory where the query result is stored.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <code>value</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>value</code> is NULL or query is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<code>value_size</code> does not match the size of the result. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <code>value</code> returns NULL. <code>value</code> is unaltered.

#### 2.6.5.4 `amd_dbgapi_status_t` `AMD_DBGAPI amd_dbgapi_process_set_progress ( amd_dbgapi_process_id_t process_id, amd_dbgapi_progress_t progress )`

Set the progress required for a process.

## Parameters

in	<code>process_id</code>	The process being controlled.
in	<code>progress</code>	The progress being set.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the progress has been set.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. The progress setting is not changed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>progress</code> is invalid. The progress setting is not changed.

## 2.7 Code Objects

Operations related to AMD GPU code objects loaded into a process.

### Data Structures

- struct `amd_dbgapi_code_object_id_t`  
*Opaque code object handle.*

### Macros

- #define `AMD_DBGAPI_CODE_OBJECT_NONE` (`amd_dbgapi_code_object_id_t{ 0 }`)  
*The NULL code object handle.*

### Enumerations

- enum `amd_dbgapi_code_object_info_t` { `AMD_DBGAPI_CODE_OBJECT_INFO_URI_NAME` = 1, `AMD_DBGAPI_CODE_OBJECT_INFO_LOAD_ADDRESS` = 2 }
- Code object queries that are supported by `amd_dbgapi_code_object_get_info`.*

### Functions

- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_code_object_get_info` (`amd_dbgapi_process_id_t` process\_id, `amd_dbgapi_code_object_id_t` code\_object\_id, `amd_dbgapi_code_object_info_t` query, `size_t` value\_size, `void *`value)  
*Query information about a code object.*
- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_code_object_list` (`amd_dbgapi_process_id_t` process\_id, `size_t *`code\_object\_count, `amd_dbgapi_code_object_id_t **`code\_objects, `amd_dbgapi_changed_t *`changed)  
*Return the list of loaded code objects for a process.*

#### 2.7.1 Detailed Description

Operations related to AMD GPU code objects loaded into a process. AMD GPU code objects are standard ELF shared libraries defined in [User Guide for AMDGPU Backend - Code Object](#).

AMD GPU code objects can be embedded in the host executable code object that is loaded into memory or be in a separate file in the file system. The AMD GPU loader supports loading either from memory or from files. The loader selects the segments to put into memory that contain the code and data necessary for AMD GPU code execution. It allocates global memory to map these segments and performs necessary relocations to create the loaded code object.

#### 2.7.2 Macro Definition Documentation

##### 2.7.2.1 #define AMD\_DBGAPI\_CODE\_OBJECT\_NONE (amd\_dbgapi\_code\_object\_id\_t{ 0 })

The NULL code object handle.

### 2.7.3 Enumeration Type Documentation

#### 2.7.3.1 enum amd\_dbgapi\_code\_object\_info\_t

Code object queries that are supported by [amd\\_dbgapi\\_code\\_object\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_code\\_object\\_get\\_info](#).

##### Enumerator

**AMD\_DBGAPI\_CODE\_OBJECT\_INFO\_URI\_NAME** The URI name from which the code object was loaded. The type of this attribute is a NUL terminated `char*`. The URI name syntax is defined by the following BNF syntax:

```
uri ::= file_uri
file_uri ::= "file://" file_path [ "#" ( file_slice | embedded ) ]
file_slice ::= "offset=" number "&" "size=" number
embedded ::= "embedded=" name "&" "arch=" name
file_path ::= URI_ENCODED_OS_FILE_PATH
number ::= HEX_NUMBER | DECIMAL_NUMBER
name ::= URI_ENCODED_IDENTIFIER
```

If the code object is loaded from memory, then the process file system can be used. For example, on Linux `/proc/123/mem#offset=0x2000,size=0x1000`.

It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

**AMD\_DBGAPI\_CODE\_OBJECT\_INFO\_LOAD\_ADDRESS** The address at which the code object is loaded in global memory. The type of this attributes is [amd\\_dbgapi\\_global\\_address\\_t](#).

### 2.7.4 Function Documentation

**2.7.4.1** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_code_object_get_info ( amd_dbgapi_process_id_t process_id, amd_dbgapi_code_object_id_t code_object_id, amd_dbgapi_code_object_info_t query, size_t value_size, void * value )`

Query information about a code object.

[amd\\_dbgapi\\_code\\_object\\_info\\_t](#) specifies the queries supported and the type returned using the `value` argument.

##### Parameters

in	<i>process_id</i>	The process to which the code object belongs.
in	<i>code_object_id</i>	The handle of the code object being queried.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <code>value</code> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

##### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <code>value</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>value</code> is unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	process_id is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_CODE_OBJECT_ID</a>	code_object_id is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	value is NULL or query is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	value_size does not match the size of the result. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate value returns NULL. value is unaltered.

2.7.4.2 **amd\_dbgapi\_status\_t** **AMD\_DBGAPI** amd\_dbgapi\_code\_object\_list ( **amd\_dbgapi\_process\_id\_t** process\_id, size\_t \* code\_object\_count, **amd\_dbgapi\_code\_object\_id\_t** \*\* code\_objects, **amd\_dbgapi\_changed\_t** \* changed )

Return the list of loaded code objects for a process.

The order of the code object handles in the list is unspecified and can vary between calls.

#### Parameters

in	process_id	The process for which the code object list is requested.
out	code_object_count	The number of code objects currently loaded.
out	code_objects	If changed is not NULL and the code object list has not changed since the last call to <a href="#">amd_dbgapi_code_object_list</a> then return NULL. Otherwise, return a pointer to an array of <a href="#">amd_dbgapi_code_object_id_t</a> with code_object_count elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.
in,out	changed	If NULL then left unaltered. If non-NULL, set to <a href="#">AMD_DBGAPI_CHANGED_NO</a> if the list of code objects is the same as when <a href="#">amd_dbgapi_code_object_list</a> was last called, otherwise set to <a href="#">AMD_DBGAPI_CHANGED_YES</a> .

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in changed, code_object_count, and code_objects.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and code_object_count, code_objects, and changed are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and code_object_count, code_objects, and changed are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	process_id is invalid. code_object_count, code_objects, and changed are unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>code_object_count</code> or <code>code_objects</code> are NULL, or <code>changed</code> is invalid. <code>code_object_count</code> , <code>code_objects</code> , and <code>changed</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <code>code_objects</code> returns NULL. <code>code_object_count</code> , <code>code_objects</code> , and <code>changed</code> are unaltered.

## 2.8 Agents

Operations related to AMD GPU agents accessible to a process.

### Data Structures

- struct `amd_dbgapi_agent_id_t`

*Opaque agent handle.*

### Macros

- `#define AMD_DBGAPI_AGENT_NONE (amd_dbgapi_agent_id_t{ 0 })`

*The NULL agent handle.*

### Enumerations

- enum `amd_dbgapi_agent_info_t` {  
`AMD_DBGAPI_AGENT_INFO_NAME = 1, AMD_DBGAPI_AGENT_INFO_ARCHITECTURE = 2, AMD_DBGAPI_AGENT_INFO_PCIE_SLOT = 3, AMD_DBGAPI_AGENT_INFO_PCIE_VENDOR_ID = 4,`  
`AMD_DBGAPI_AGENT_INFO_PCIE_DEVICE_ID = 5, AMD_DBGAPI_AGENT_INFO_SHADER_ENGINE_COUNT = 6, AMD_DBGAPI_AGENT_INFO_COMPUTE_UNIT_COUNT = 7, AMD_DBGAPI_AGENT_INFO_NUM_SIMD_PER_COMPUTE_UNIT = 8,`  
`AMD_DBGAPI_AGENT_INFO_MAX_WAVES_PER_SIMD = 9 }`

*Agent queries that are supported by `amd_dbgapi_agent_get_info`.*

### Functions

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_agent_get_info (amd_dbgapi_process_id_t process_id, amd_dbgapi_agent_id_t agent_id, amd_dbgapi_agent_info_t query, size_t value_size, void *value)`  
*Query information about an agent.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_agent_list (amd_dbgapi_process_id_t process_id, size_t *agent_count, amd_dbgapi_agent_id_t **agents, amd_dbgapi_changed_t *changed)`  
*Return the list of agents for a process.*

#### 2.8.1 Detailed Description

Operations related to AMD GPU agents accessible to a process. Agent is the term for AMD GPU devices that can be accessed by the process.

#### 2.8.2 Macro Definition Documentation

##### 2.8.2.1 `#define AMD_DBGAPI_AGENT_NONE (amd_dbgapi_agent_id_t{ 0 })`

The NULL agent handle.



### 2.8.3 Enumeration Type Documentation

#### 2.8.3.1 enum amd\_dbgapi\_agent\_info\_t

Agent queries that are supported by [amd\\_dbgapi\\_agent\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_agent\\_get\\_info](#).

##### Enumerator

**AMD\_DBGAPI\_AGENT\_INFO\_NAME** Agent name. The type of this attribute is a pointer to a NUL terminated `char*`. It is allocated by [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) and is owned by the client.

**AMD\_DBGAPI\_AGENT\_INFO\_ARCHITECTURE** Return the architecture of this agent. The type of this attribute is [amd\\_dbgapi\\_architecture\\_id\\_t](#).

**AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_SLOT** PCIe slot of the agent in BDF format (see [Bus:Device.Function (B-D-F) Notation][bdf]). The type of this attribute is `uint16_t`.

**AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_VENDOR\_ID** PCIe vendor ID of the agent. The type of this attribute is `uint32_t`.

**AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_DEVICE\_ID** PCIe device ID of the agent. The type of this attribute is `uint32_t`.

**AMD\_DBGAPI\_AGENT\_INFO\_SHADER\_ENGINE\_COUNT** The number of Shader Engines (SE) in the agent. The type of this attribute is `size_t`.

**AMD\_DBGAPI\_AGENT\_INFO\_COMPUTE\_UNIT\_COUNT** Number of compute units available in the agent. The type of this attribute is `size_t`.

**AMD\_DBGAPI\_AGENT\_INFO\_NUM\_SIMD\_PER\_COMPUTE\_UNIT** Number of SIMDs per compute unit (CU). The type of this attribute is `size_t`.

**AMD\_DBGAPI\_AGENT\_INFO\_MAX\_WAVES\_PER\_SIMD** Maximum number of waves possible in a SIMD. The type of this attribute is `size_t`.

### 2.8.4 Function Documentation

#### 2.8.4.1 amd\_dbgapi\_status\_t AMD\_DBGAPI amd\_dbgapi\_agent\_get\_info ( amd\_dbgapi\_process\_id\_t process\_id, amd\_dbgapi\_agent\_id\_t agent\_id, amd\_dbgapi\_agent\_info\_t query, size\_t value\_size, void \* value )

Query information about an agent.

[amd\\_dbgapi\\_agent\\_info\\_t](#) specifies the queries supported and the type returned using the `value` argument.

##### Parameters

in	<i>process_id</i>	The process to which the agent belongs.
in	<i>agent_id</i>	The handle of the agent being queried.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <code>value</code> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

##### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <code>value</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_AGENT_ID</a>	<code>agent_id</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>value</code> is NULL or <code>query</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<code>value_size</code> does not match the size of the result. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <code>value</code> returns NULL. <code>value</code> is unaltered.

**2.8.4.2** `amd_dbgapi_status_t` **AMD\_DBGAPI** `amd_dbgapi_agent_list ( amd_dbgapi_process_id_t process_id, size_t * agent_count, amd_dbgapi_agent_id_t ** agents, amd_dbgapi_changed_t * changed )`

Return the list of agents for a process.

The order of the agent handles in the list is unspecified and can vary between calls.

#### Parameters

in	<code>process_id</code>	The process for which the agent list is requested.
out	<code>agent_count</code>	The number of agents accessed by the process.
out	<code>agents</code>	If <code>changed</code> is not NULL and the agent list has not changed since the last call to <a href="#">amd_dbgapi_agent_list</a> then return NULL. Otherwise, return a pointer to an array of <a href="#">amd_dbgapi_agent_id_t</a> with <code>agent_count</code> elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.
in, out	<code>changed</code>	If NULL then left unaltered. If non-NULL, set to <a href="#">AMD_DBGAPI_CHANGED_NO</a> if the list of agents is the same as when <a href="#">amd_dbgapi_agent_list</a> was last called, otherwise set to <a href="#">AMD_DBGAPI_CHANGED_YES</a> .

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <code>changed</code> , <code>agent_count</code> , and <code>agents</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>agent_count</code> , <code>agents</code> , and <code>changed</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>agent_count</code> , <code>agents</code> , and <code>changed</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. <code>agent_count</code> , <code>agents</code> , and <code>changed</code> are unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	agent_count or agents are NULL, or changed is invalid. agent_count, agents, and changed are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate agents returns NULL. agent_count, agents, and changed are unaltered.

## 2.9 Queues

Operations related to AMD GPU queues.

### Data Structures

- struct `amd_dbgapi_queue_id_t`  
*Opaque queue handle.*

### Macros

- #define `AMD_DBGAPI_QUEUE_NONE` (`amd_dbgapi_queue_id_t{ 0 }`)  
*The NULL queue handle.*

### Typedefs

- typedef uint64\_t `amd_dbgapi_queue_packet_id_t`  
*Queue packet ID.*

### Enumerations

- enum `amd_dbgapi_queue_info_t` {  
    `AMD_DBGAPI_QUEUE_INFO_AGENT` = 1, `AMD_DBGAPI_QUEUE_INFO_ARCHITECTURE` = 2, `AMD_DBGAPI_QUEUE_TYPE` = 3, `AMD_DBGAPI_QUEUE_INFO_STATE` = 4,  
    `AMD_DBGAPI_QUEUE_INFO_ERROR_REASON` = 5 }  
*Queue queries that are supported by `amd_dbgapi_queue_get_info`.*
- enum `amd_dbgapi_queue_type_t` { `AMD_DBGAPI_QUEUE_TYPE_UNKNOWN` = 0, `AMD_DBGAPI_QUEUE_TYPE_HSA_KERNEL_DISPATCH_MULTIPLE_PRODUCER` = 1, `AMD_DBGAPI_QUEUE_TYPE_HSA_KERNEL_DISPATCH_SINGLE_PRODUCER` = 2, `AMD_DBGAPI_QUEUE_TYPE_AMD_PM4` = 3 }  
*Queue type.*
- enum `amd_dbgapi_queue_state_t` { `AMD_DBGAPI_QUEUE_STATE_VALID` = 1, `AMD_DBGAPI_QUEUE_STATE_ERROR` = 2 }  
*Queue state.*
- enum `amd_dbgapi_queue_error_reason_t` { `AMD_DBGAPI_QUEUE_ERROR_REASON_INVALID_PACKET` = (1 << 0), `AMD_DBGAPI_QUEUE_ERROR_REASON_MEMORY_VIOLATION` = (1 << 1), `AMD_DBGAPI_QUEUE_ERROR_REASON_ASSERT_TRAP` = (1 << 2), `AMD_DBGAPI_QUEUE_ERROR_REASON_WAVE_ERROR` = (1 << 3) }  
*A bit mask of the reasons that a queue is in error.*

### Functions

- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_queue_get_info` (`amd_dbgapi_process_id_t` process\_id, `amd_dbgapi_queue_id_t` queue\_id, `amd_dbgapi_queue_info_t` query, `size_t` value\_size, `void *`value)  
*Query information about a queue.*
- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_queue_list` (`amd_dbgapi_process_id_t` process\_id, `size_t` \*queue\_count, `amd_dbgapi_queue_id_t` \*\*queues, `amd_dbgapi_changed_t` \*changed)  
*Return the list of queues for a process.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_queue_packet_list (amd_dbgapi_process_id_t process_id, amd_dbgapi_queue_id_t queue_id, amd_dbgapi_queue_packet_id_t *first_packet_id, amd_dbgapi_size_t *packets_byte_size, void **packets_bytes)`

*Return the packets for a queue of a process.*

### 2.9.1 Detailed Description

Operations related to AMD GPU queues. Queues are user mode data structures that allow packets to be inserted that control the AMD GPU agents. The dispatch packet is used to initiate the execution of a grid of waves.

### 2.9.2 Macro Definition Documentation

2.9.2.1 `#define AMD_DBGAPI_QUEUE_NONE (amd_dbgapi_queue_id_t{ 0 })`

The NULL queue handle.

### 2.9.3 Typedef Documentation

2.9.3.1 `typedef uint64_t amd_dbgapi_queue_packet_id_t`

Queue packet ID.

The meaning of the packet ID is dependent on the queue type. See [amd\\_dbgapi\\_queue\\_type\\_t](#).

### 2.9.4 Enumeration Type Documentation

2.9.4.1 `enum amd_dbgapi_queue_error_reason_t`

A bit mask of the reasons that a queue is in error.

Enumerator

**AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_INVALID\_PACKET** A packet on the queue is invalid.

**AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_MEMORY\_VIOLATION** A wave on the queue had a memory violation.

**AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_ASSERT\_TRAP** A wave on the queue had an assert trap.

**AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_WAVE\_ERROR** A wave on the queue executed an instruction that caused an error. The [AMD\\_DBGAPI\\_WAVE\\_INFO\\_STOP\\_REASON](#) query can be used on the waves of the queue to determine the exact reason.

2.9.4.2 `enum amd_dbgapi_queue_info_t`

Queue queries that are supported by [amd\\_dbgapi\\_queue\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_queue\\_get\\_info](#).

Enumerator

**AMD\_DBGAPI\_QUEUE\_INFO\_AGENT** Return the agent to which this queue belongs. The type of this attribute is [amd\\_dbgapi\\_agent\\_id\\_t](#).

**AMD\_DBGAPI\_QUEUE\_INFO\_ARCHITECTURE** Return the architecture of this queue. The type of this attribute is [amd\\_dbgapi\\_architecture\\_id\\_t](#).

**AMD\_DBGAPI\_QUEUE\_TYPE** Return the queue type. The type of this attribute is `uint32_t` with values from [amd\\_dbgapi\\_queue\\_type\\_t](#).

**AMD\_DBGAPI\_QUEUE\_INFO\_STATE** Return the queue state. The type of this attribute is `uint32_t` with values from [amd\\_dbgapi\\_queue\\_state\\_t](#).

**AMD\_DBGAPI\_QUEUE\_INFO\_ERROR\_REASON** Return the reason the queue is in error as a bit set. The type of this attribute is `uint64_t` with values defined by [amd\\_dbgapi\\_queue\\_error\\_reason\\_t](#).

#### 2.9.4.3 enum amd\_dbgapi\_queue\_state\_t

Queue state.

Enumerator

**AMD\_DBGAPI\_QUEUE\_STATE\_VALID** Queue is in a valid state.

**AMD\_DBGAPI\_QUEUE\_STATE\_ERROR** Queue is in an error state. When a queue enters the error state, a wave stop event will be created for all non-stopped waves. All waves of the queue will include the [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_QUEUE\\_ERROR](#) stop reason.

#### 2.9.4.4 enum amd\_dbgapi\_queue\_type\_t

Queue type.

Indicates which queue mechanic is supported by the queue.

Enumerator

**AMD\_DBGAPI\_QUEUE\_TYPE\_UNKNOWN** Unknown queue type.

**AMD\_DBGAPI\_QUEUE\_TYPE\_HSA\_KERNEL\_DISPATCH\_MULTIPLE\_PRODUCER** Queue supports the HSA A kernel dispatch with multiple producers protocol. This follows the multiple producers mechanics described by [HSA Platform System Architecture Specification: Requirement: User mode queuing](#) and uses the HSA Architected Queuing Language (AQL) packet format described in [HSA Platform System Architecture Specification: Requirement: Architected Queuing Language \(AQL\)](#).

For this queue type the AQL dispatch ID is used for [amd\\_dbgapi\\_queue\\_packet\\_id\\_t](#).

**AMD\_DBGAPI\_QUEUE\_TYPE\_HSA\_KERNEL\_DISPATCH\_SINGLE\_PRODUCER** Queue supports the HSA kernel dispatch with single producer protocol. This follows the single producer mechanics described by [HSA Platform System Architecture Specification: Requirement: User mode queuing](#) and uses the HSA Architected Queuing Language (AQL) packet format described in [HSA Platform System Architecture Specification: Requirement: Architected Queuing Language \(AQL\)](#).

For this queue type the AQL dispatch ID is used for [amd\\_dbgapi\\_queue\\_packet\\_id\\_t](#). It is only unique within a single queue of a single process.

**AMD\_DBGAPI\_QUEUE\_TYPE\_AMD\_PM4** Queue supports the AMD PM4 protocol.

### 2.9.5 Function Documentation

2.9.5.1 `amd_dbgapi_status_t` AMD\_DBGAPI `amd_dbgapi_queue_get_info` ( `amd_dbgapi_process_id_t` *process\_id*,  
`amd_dbgapi_queue_id_t` *queue\_id*, `amd_dbgapi_queue_info_t` *query*, `size_t` *value\_size*, `void *` *value* )

Query information about a queue.

[amd\\_dbgapi\\_queue\\_info\\_t](#) specifies the queries supported and the type returned using the `value` argument.

## Parameters

in	<i>process_id</i>	The process to which the queue belongs.
in	<i>queue_id</i>	The handle of the queue being queried.
in	<i>query</i>	The query being requested.
out	<i>value</i>	Pointer to memory where the query result is stored.
in	<i>value_size</i>	Size of the memory pointed to by <i>value</i> . Must be equal to the byte size of the query result.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>value</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_QUEUE_ID</a>	<i>queue_id</i> is invalid. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>value</i> is NULL or <i>query</i> is invalid. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<i>value_size</i> does not match the size of the result. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <i>value</i> returns NULL. <i>value</i> is unaltered.

**2.9.5.2** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_queue_list ( amd_dbgapi_process_id_t process_id, size_t * queue_count, amd_dbgapi_queue_id_t ** queues, amd_dbgapi_changed_t * changed )`

Return the list of queues for a process.

The order of the queue handles in the list is unspecified and can vary between calls.

## Parameters

in	<i>process_id</i>	The process for which the queue list is requested.
out	<i>queue_count</i>	The number of queues accessed by the process.
out	<i>queues</i>	If <i>changed</i> is not NULL and the queue list has not changed since the last call to <a href="#">amd_dbgapi_queue_list</a> then return NULL. Otherwise, return a pointer to an array of <a href="#">amd_dbgapi_queue_id_t</a> with <i>queue_count</i> elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.



in, out	<i>changed</i>	If NULL then left unaltered. If non-NULL, set to <a href="#">AMD_DBGAPI_CHANGED_NO</a> if the list of queues is the same as when <a href="#">amd_dbgapi_queue_list</a> was last called, otherwise set to <a href="#">AMD_DBGAPI_CHANGED_YES</a> .
---------	----------------	---

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>changed</i> , <i>queue_count</i> , and <i>queues</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>queue_count</i> , <i>queues</i> , and <i>changed</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>queue_count</i> , <i>queues</i> , and <i>changed</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. <i>queue_count</i> , <i>queues</i> , and <i>changed</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>queue_count</i> or <i>queues</i> are NULL, or <i>changed</i> is invalid. <i>queue_count</i> , <i>queues</i> , and <i>changed</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <i>queues</i> returns NULL. <i>queue_count</i> , <i>queues</i> , and <i>changed</i> are unaltered.

**2.9.5.3** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_queue_packet_list ( amd_dbgapi_process_id_t process_id, amd_dbgapi_queue_id_t queue_id, amd_dbgapi_queue_packet_id_t * first_packet_id, amd_dbgapi_size_t * packets_byte_size, void ** packets_bytes )`

Return the packets for a queue of a process.

Since the AMD GPU is asynchronously reading the packets this is only a snapshot of the packets present in the queue, and only includes the packets that the producer has made available to the queue. In obtaining the snapshot the library may pause the queue processing in order to get a consistent snapshot.

The queue packets are returned as a byte block that the client must interpret according to the packet ABI determined by the queue type available using the [AMD\\_DBGAPI\\_QUEUE\\_TYPE](#) query. See [amd\\_dbgapi\\_queue\\_type\\_t](#).

## Parameters

in	<i>process_id</i>	The process of the queue for which the packet list is requested.
in	<i>queue_id</i>	The queue for which the packet list is requested.
out	<i>first_packet_id</i>	The packet ID for the first packet in <i>packets_bytes</i> . If <i>packets_byte_size</i> is zero, then the packet ID for the next packet added to the queue.
out	<i>packets_byte_size</i>	The number of bytes of packets on the queue.
out	<i>packets_bytes</i>	A pointer to an array of <i>packets_byte_size</i> bytes. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>packets_byte_size</i> and <i>packets_bytes</i> .
---	---

<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized and <code>packets_byte_size</code> and <code>packets_bytes</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and <code>packets_byte_size</code> and <code>packets_bytes</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</i></a>	<code>process_id</code> is invalid. <code>packets_byte_size</code> and <code>packets_bytes</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</i></a>	<code>packets_byte_size</code> or <code>packets_bytes</code> are NULL. <code>packets_byte_size</code> and <code>packets_bytes</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</i></a>	This will be reported if the <a href="#"><code>amd_dbgapi_callbacks_s::allocate_memory</code></a> callback used to allocate <code>packets_bytes</code> returns NULL. <code>packets_byte_size</code> and <code>packets_bytes</code> are unaltered.

## 2.10 Dispatches

Operations related to AMD GPU dispatches.

### Data Structures

- struct `amd_dbgapi_dispatch_id_t`  
*Opaque dispatch handle.*

### Macros

- #define `AMD_DBGAPI_DISPATCH_NONE` (`amd_dbgapi_dispatch_id_t`{ 0 })  
*The NULL dispatch handle.*

### Enumerations

- enum `amd_dbgapi_dispatch_info_t` {  
`AMD_DBGAPI_DISPATCH_INFO_QUEUE` = 1, `AMD_DBGAPI_DISPATCH_INFO_AGENT` = 2, `AMD_DBGAPI_DISPATCH_INFO_ARCHITECTURE` = 3, `AMD_DBGAPI_DISPATCH_INFO_PACKET_ID` = 4,  
`AMD_DBGAPI_DISPATCH_INFO_BARRIER` = 5, `AMD_DBGAPI_DISPATCH_INFO_ACQUIRE_FENCE` = 6, `AMD_DBGAPI_DISPATCH_INFO_RELEASE_FENCE` = 7, `AMD_DBGAPI_DISPATCH_INFO_GRID_DIMENSIONS` = 8,  
`AMD_DBGAPI_DISPATCH_INFO_WORK_GROUP_SIZES` = 9, `AMD_DBGAPI_DISPATCH_INFO_GRID_SIZES` = 10, `AMD_DBGAPI_DISPATCH_INFO_PRIVATE_SEGMENT_SIZE` = 11, `AMD_DBGAPI_DISPATCH_INFO_GROUP_SEGMENT_SIZE` = 12,  
`AMD_DBGAPI_DISPATCH_INFO_KERNEL_ARGUMENT_SEGMENT_ADDRESS` = 13, `AMD_DBGAPI_DISPATCH_INFO_KERNEL_ENTRY_ADDRESS` = 14 }  
*Dispatch queries that are supported by `amd_dbgapi_dispatch_get_info`.*
- enum `amd_dbgapi_dispatch_barrier_t` { `AMD_DBGAPI_DISPATCH_BARRIER_NONE` = 0, `AMD_DBGAPI_DISPATCH_BARRIER_PRESENT` = 1 }  
*Dispatch barrier.*
- enum `amd_dbgapi_dispatch_fence_scope_t` { `AMD_DBGAPI_DISPATCH_FENCE_SCOPE_NONE` = 0, `AMD_DBGAPI_DISPATCH_FENCE_SCOPE_AGENT` = 1, `AMD_DBGAPI_DISPATCH_FENCE_SCOPE_SYSTEM` = 2 }  
*Dispatch memory fence scope.*

### Functions

- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_dispatch_get_info` (`amd_dbgapi_process_id_t` process\_id, `amd_dbgapi_dispatch_id_t` dispatch\_id, `amd_dbgapi_dispatch_info_t` query, `size_t` value\_size, void \*value)  
*Query information about a dispatch.*
- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_dispatch_list` (`amd_dbgapi_process_id_t` process\_id, `size_t` \*dispatch\_count, `amd_dbgapi_dispatch_id_t` \*\*dispatches, `amd_dbgapi_changed_t` \*changed)  
*Return the list of dispatches for a process.*

#### 2.10.1 Detailed Description

Operations related to AMD GPU dispatches. Dispatches are initiated by queue dispatch packets in the format supported by the queue. See `amd_dbgapi_queue_type_t`. Dispatches are the means that waves are created on the AMD GPU.

## 2.10.2 Macro Definition Documentation

### 2.10.2.1 `#define AMD_DBGAPI_DISPATCH_NONE (amd_dbgapi_dispatch_id_t{ 0 })`

The NULL dispatch handle.

## 2.10.3 Enumeration Type Documentation

### 2.10.3.1 `enum amd_dbgapi_dispatch_barrier_t`

Dispatch barrier.

Controls when the dispatch will start being executed relative to previous packets on the queue.

Enumerator

**`AMD_DBGAPI_DISPATCH_BARRIER_NONE`** Dispatch has no barrier.

**`AMD_DBGAPI_DISPATCH_BARRIER_PRESENT`** Dispatch has a barrier. The dispatch will not be executed until all proceeding packets on the queue have completed.

### 2.10.3.2 `enum amd_dbgapi_dispatch_fence_scope_t`

Dispatch memory fence scope.

Controls how memory is acquired before a dispatch starts executing and released after the dispatch completes execution.

Enumerator

**`AMD_DBGAPI_DISPATCH_FENCE_SCOPE_NONE`** There is no fence.

**`AMD_DBGAPI_DISPATCH_FENCE_SCOPE_AGENT`** There is a fence with agent memory scope.

**`AMD_DBGAPI_DISPATCH_FENCE_SCOPE_SYSTEM`** There is a fence with system memory scope.

### 2.10.3.3 `enum amd_dbgapi_dispatch_info_t`

Dispatch queries that are supported by [amd\\_dbgapi\\_dispatch\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_queue\\_get\\_info](#).

Enumerator

**`AMD_DBGAPI_DISPATCH_INFO_QUEUE`** Return the queue to which this dispatch belongs. The type of this attribute is [amd\\_dbgapi\\_queue\\_id\\_t](#).

**`AMD_DBGAPI_DISPATCH_INFO_AGENT`** Return the agent to which this queue belongs. The type of this attribute is [amd\\_dbgapi\\_agent\\_id\\_t](#).

**`AMD_DBGAPI_DISPATCH_INFO_ARCHITECTURE`** Return the architecture of this dispatch. The type of this attribute is [amd\\_dbgapi\\_architecture\\_id\\_t](#).

**`AMD_DBGAPI_DISPATCH_INFO_PACKET_ID`** Return the queue packet ID of the dispatch packet that initiated the dispatch. The type of this attribute is [amd\\_dbgapi\\_queue\\_packet\\_id\\_t](#).

**`AMD_DBGAPI_DISPATCH_INFO_BARRIER`** Return the dispatch barrier setting. The type of this attribute is [uint32\\_t](#) with values defined by [amd\\_dbgapi\\_dispatch\\_barrier\\_t](#).

**AMD\_DBGAPI\_DISPATCH\_INFO\_ACQUIRE\_FENCE** Return the dispatch acquire fence. The type of this attribute is `uint32_t` with values defined by `amd_dbgapi_dispatch_fence_scope_t`.

**AMD\_DBGAPI\_DISPATCH\_INFO\_RELEASE\_FENCE** Return the dispatch release fence. The type of this attribute is `uint32_t` with values defined by `amd_dbgapi_dispatch_fence_scope_t`.

**AMD\_DBGAPI\_DISPATCH\_INFO\_GRID\_DIMENSIONS** Return the dispatch grid dimensionality. The type of this attribute is `uint32` with a value of 1, 2, or 3.

**AMD\_DBGAPI\_DISPATCH\_INFO\_WORK\_GROUP\_SIZES** Return the dispatch workgroup size (work-items) in the X, Y, and Z dimensions. The type of this attribute is `uint16_t[3]`.

**AMD\_DBGAPI\_DISPATCH\_INFO\_GRID\_SIZES** Return the dispatch grid size (work-items) in the X, Y, and Z dimensions. The type of this attribute is `uint32_t[3]`.

**AMD\_DBGAPI\_DISPATCH\_INFO\_PRIVATE\_SEGMENT\_SIZE** Return the dispatch private segment size in bytes. The type of this attribute is `amd_dbgapi_size_t`.

**AMD\_DBGAPI\_DISPATCH\_INFO\_GROUP\_SEGMENT\_SIZE** Return the dispatch group segment size in bytes. The type of this attribute is `amd_dbgapi_size_t`.

**AMD\_DBGAPI\_DISPATCH\_INFO\_KERNEL\_ARGUMENT\_SEGMENT\_ADDRESS** Return the dispatch kernel argument segment address. The type of this attribute is `amd_dbgapi_global_address_t`.

**AMD\_DBGAPI\_DISPATCH\_INFO\_KERNEL\_ENTRY\_ADDRESS** Return the dispatch kernel function address. The type of this attribute is `amd_dbgapi_global_address_t`.

## 2.10.4 Function Documentation

2.10.4.1 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_dispatch_get_info ( amd_dbgapi_process_id_t process_id, amd_dbgapi_dispatch_id_t dispatch_id, amd_dbgapi_dispatch_info_t query, size_t value_size, void * value )`

Query information about a dispatch.

`amd_dbgapi_dispatch_info_t` specifies the queries supported and the type returned using the `value` argument.

### Parameters

in	<code>process_id</code>	The process to which the queue belongs.
in	<code>dispatch_id</code>	The handle of the dispatch being queried.
in	<code>query</code>	The query being requested.
in	<code>value_size</code>	Size of the memory pointed to by <code>value</code> . Must be equal to the byte size of the query result.
out	<code>value</code>	Pointer to memory where the query result is stored.

### Return values

<code>AMD_DBGAPI_STATUS_SUCCESS</code>	The function has been executed successfully and the result is stored in <code>value</code> .
<code>AMD_DBGAPI_STATUS_FATAL</code>	A fatal error occurred. The library is left uninitialized and <code>value</code> is unaltered.
<code>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</code>	The library is not initialized. The library is left uninitialized and <code>value</code> is unaltered.
<code>AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</code>	<code>process_id</code> is invalid. <code>value</code> is unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_DISPATCH_ID</a>	queue_id is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	value is NULL or query is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	value_size does not match the size of the result. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate value returns NULL. value is unaltered.

2.10.4.2 **amd\_dbgapi\_status\_t** [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_dispatch\\_list](#) ( [amd\\_dbgapi\\_process\\_id\\_t](#) *process\_id*, [size\\_t](#)\* *dispatch\_count*, [amd\\_dbgapi\\_dispatch\\_id\\_t](#)\*\* *dispatches*, [amd\\_dbgapi\\_changed\\_t](#)\* *changed* )

Return the list of dispatches for a process.

The order of the dispatch handles in the list is unspecified and can vary between calls.

#### Parameters

in	<i>process_id</i>	The process for which the dispatch list is requested.
out	<i>dispatch_count</i>	The number of dispatches active for a process.
out	<i>dispatches</i>	If <i>changed</i> is not NULL and the dispatch list has not changed since the last call to <a href="#">amd_dbgapi_dispatch_list</a> then return NULL. Otherwise, return a pointer to an array of <a href="#">amd_dbgapi_dispatch_id_t</a> with <i>dispatch_count</i> elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.
in, out	<i>changed</i>	If NULL then left unaltered. If non-NULL, set to <a href="#">AMD_DBGAPI_CHANGED_NO</a> if the list of agents is the same as when <a href="#">amd_dbgapi_agent_list</a> was last called, otherwise set to <a href="#">AMD_DBGAPI_CHANGED_YES</a> .

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>changed</i> , <i>dispatch_count</i> , and <i>dispatches</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>changed</i> , <i>dispatch_count</i> , and <i>dispatches</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>changed</i> , <i>dispatch_count</i> , and <i>dispatches</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. <i>dispatch_count</i> , <i>dispatches</i> , and <i>changed</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>dispatch_count</i> or <i>dispatches</i> are NULL, or <i>changed</i> is invalid. <i>dispatch_count</i> , <i>dispatches</i> , and <i>changed</i> are unaltered.

## 2.11 Wave

Operations related to AMD GPU waves.

### Data Structures

- struct `amd_dbgapi_wave_id_t`

*Opaque wave handle.*

### Macros

- `#define AMD_DBGAPI_WAVE_NONE (amd_dbgapi_wave_id_t{ 0 })`

*The NULL wave handle.*

### Enumerations

- enum `amd_dbgapi_wave_info_t` {  
`AMD_DBGAPI_WAVE_INFO_STATE = 1, AMD_DBGAPI_WAVE_INFO_STOP_REASON = 2, AMD_DBGAPI_WAVE_INFO_WATCHPOINTS = 3, AMD_DBGAPI_WAVE_INFO_DISPATCH = 4,`  
`AMD_DBGAPI_WAVE_INFO_QUEUE = 5, AMD_DBGAPI_WAVE_INFO_AGENT = 6, AMD_DBGAPI_WAVE_INFO_ARCHITECTURE = 7, AMD_DBGAPI_WAVE_INFO_PC = 8,`  
`AMD_DBGAPI_WAVE_INFO_EXEC_MASK = 9, AMD_DBGAPI_WAVE_INFO_WORK_GROUP_COORD = 10,`  
`AMD_DBGAPI_WAVE_INFO_WAVE_NUMBER_IN_WORK_GROUP = 11, AMD_DBGAPI_WAVE_INFO_LANE_COUNT = 12 }`

*Wave queries that are supported by `amd_dbgapi_wave_get_info`.*

- enum `amd_dbgapi_wave_state_t` { `AMD_DBGAPI_WAVE_STATE_RUN = 1, AMD_DBGAPI_WAVE_STATE_SINGLE_STEP = 2, AMD_DBGAPI_WAVE_STATE_STOP = 3 }`

*The execution state of a wave.*

- enum `amd_dbgapi_wave_stop_reason_t` {  
`AMD_DBGAPI_WAVE_STOP_REASON_NONE = 0, AMD_DBGAPI_WAVE_STOP_REASON_BREAKPOINT = (1 << 0), AMD_DBGAPI_WAVE_STOP_REASON_WATCHPOINT = (1 << 1), AMD_DBGAPI_WAVE_STOP_REASON_SINGLE_STEP = (1 << 2),`  
`AMD_DBGAPI_WAVE_STOP_REASON_QUEUE_ERROR = (1 << 3), AMD_DBGAPI_WAVE_STOP_REASON_FP_INPUT_DENORMAL = (1 << 4), AMD_DBGAPI_WAVE_STOP_REASON_FP_DIVIDE_BY_0 = (1 << 5),`  
`AMD_DBGAPI_WAVE_STOP_REASON_FP_OVERFLOW = (1 << 6), AMD_DBGAPI_WAVE_STOP_REASON_FP_UNDERFLOW = (1 << 7), AMD_DBGAPI_WAVE_STOP_REASON_FP_INEXACT = (1 << 8),`  
`AMD_DBGAPI_WAVE_STOP_REASON_FP_INVALID_OPERATION = (1 << 9), AMD_DBGAPI_WAVE_STOP_REASON_INT_DIVIDE_BY_0 = (1 << 10),`  
`AMD_DBGAPI_WAVE_STOP_REASON_DEBUG_TRAP = (1 << 11), AMD_DBGAPI_WAVE_STOP_REASON_ASSERT_TRAP = (1 << 12), AMD_DBGAPI_WAVE_STOP_REASON_TRAP = (1 << 13),`  
`AMD_DBGAPI_WAVE_STOP_REASON_MEMORY_VIOLATION = (1 << 14),`  
`AMD_DBGAPI_WAVE_STOP_REASON_ILLEGAL_INSTRUCTION = (1 << 15), AMD_DBGAPI_WAVE_STOP_REASON_ECC_ERROR = (1 << 16),`  
`AMD_DBGAPI_WAVE_STOP_REASON_FATAL_HALT = (1 << 17), AMD_DBGAPI_WAVE_STOP_REASON_XNACK_ERROR = (1 << 18) }`

*A bit mask of the reasons that a wave stopped.*

- enum `amd_dbgapi_resume_mode_t` { `AMD_DBGAPI_RESUME_MODE_NORMAL = 0, AMD_DBGAPI_RESUME_MODE_SINGLE_STEP = 1 }`

*The mode in which to resuming the execution of a wave.*

## Functions

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_wave\\_get\\_info](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_wave\\_info\\_t](#) query, [size\\_t](#) value\_size, void \*value)  
*Query information about a wave.*
- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_wave\\_list](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [size\\_t](#) \*wave\_count, [amd\\_dbgapi\\_wave\\_id\\_t](#) \*\*waves, [amd\\_dbgapi\\_changed\\_t](#) \*changed)  
*Return the list of existing waves for a process.*
- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_wave\\_stop](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id)  
*Request a wave to stop executing.*
- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_wave\\_resume](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_resume\\_mode\\_t](#) resume\_mode)  
*Resume execution of a stopped wave.*

### 2.11.1 Detailed Description

Operations related to AMD GPU waves.

### 2.11.2 Macro Definition Documentation

#### 2.11.2.1 `#define AMD_DBGAPI_WAVE_NONE (amd_dbgapi_wave_id_t{ 0 })`

The NULL wave handle.

### 2.11.3 Enumeration Type Documentation

#### 2.11.3.1 `enum amd_dbgapi_resume_mode_t`

The mode in which to resuming the execution of a wave.

Enumerator

**`AMD_DBGAPI_RESUME_MODE_NORMAL`** Resume normal execution.

**`AMD_DBGAPI_RESUME_MODE_SINGLE_STEP`** Resume execution in in single step mode.

#### 2.11.3.2 `enum amd_dbgapi_wave_info_t`

Wave queries that are supported by [amd\\_dbgapi\\_wave\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_wave\\_get\\_info](#).

Enumerator

**`AMD_DBGAPI_WAVE_INFO_STATE`** Return the wave's state. The type of this attribute is `uint32_t` with values define by [amd\\_dbgapi\\_wave\\_state\\_t](#).

**`AMD_DBGAPI_WAVE_INFO_STOP_REASON`** Return the reason the wave stopped as a bit set. The type of this attribute is `uint64_t` with values defined by [amd\\_dbgapi\\_wave\\_stop\\_reason\\_t](#).



**AMD\_DBGAPI\_WAVE\_INFO\_WATCHPOINTS** Return the watchpoint(s) the wave triggered as a bit set. The type of this attribute is `uint64_t` with the least significant bit 1 if the watchpoint with a `amd_dbgapi_watchpoint_id_t` value of 0 was triggered and so forth. The agent of the triggered watchpoint(s) is the agent of the wave.

**AMD\_DBGAPI\_WAVE\_INFO\_DISPATCH** Return the dispatch to which this wave belongs. The type of this attribute is `amd_dbgapi_dispatch_id_t`.

**AMD\_DBGAPI\_WAVE\_INFO\_QUEUE** Return the queue to which this wave belongs. The type of this attribute is `amd_dbgapi_queue_id_t`.

**AMD\_DBGAPI\_WAVE\_INFO\_AGENT** Return the agent to which this wave belongs. The type of this attribute is `amd_dbgapi_agent_id_t`.

**AMD\_DBGAPI\_WAVE\_INFO\_ARCHITECTURE** Return the architecture of this wave. The type of this attribute is `amd_dbgapi_architecture_id_t`.

**AMD\_DBGAPI\_WAVE\_INFO\_PC** Return the current program counter value of the wave. The type of this attribute is `amd_dbgapi_global_address_t`.

**AMD\_DBGAPI\_WAVE\_INFO\_EXEC\_MASK** Return the current execution mask of the wave. Each bit of the mask maps to a lane with the least significant bit corresponding to the lane with a `amd_dbgapi_lane_id_t` value of 0 and so forth. If the bit is 1 then the lane is active, otherwise the lane is not active. The type of this attribute is `uint64_t`.

**AMD\_DBGAPI\_WAVE\_INFO\_WORK\_GROUP\_COORD** The wave workgroup coordinate in the dispatch grid dimensions. The type of this attribute is `uint32_t[3]` with elements 1, 2, and 3 corresponding to the X, Y, and Z coordinates respectively.

**AMD\_DBGAPI\_WAVE\_INFO\_WAVE\_NUMBER\_IN\_WORK\_GROUP** The wave's number in the workgroup. The type of this attribute is `uint32_t`. The work-items of a workgroup are mapped to the lanes of the waves of the workgroup in flattened work-item ID order, with the first work-item corresponding to lane 0 of wave 0, and so forth.

**AMD\_DBGAPI\_WAVE\_INFO\_LANE\_COUNT** The number of lanes supported by the wave. The type of this attribute is `amd_dbgapi_lane_id_t`.

### 2.11.3.3 enum amd\_dbgapi\_wave\_state\_t

The execution state of a wave.

Enumerator

**AMD\_DBGAPI\_WAVE\_STATE\_RUN** The wave is running.

**AMD\_DBGAPI\_WAVE\_STATE\_SINGLE\_STEP** The wave is running in single-step mode. It will execute a single instruction and then stop.

**AMD\_DBGAPI\_WAVE\_STATE\_STOP** The wave is stopped. Note that a wave may stop at any time due to the instructions it executes or because the queue it is executing on enters the error state. This will cause a `AMD_DBGAPI_EVENT_KIND_WAVE_STOP` event to be created. However, until `amd_dbgapi_next_pending_event` returns the event, the wave will continue to be reported as in the `AMD_DBGAPI_WAVE_STATE_RUN` state. Only when the `AMD_DBGAPI_EVENT_KIND_WAVE_STOP` event is returned by `amd_dbgapi_next_pending_event` will the wave will be reported in the `AMD_DBGAPI_WAVE_STATE_STOP` state.

### 2.11.3.4 enum amd\_dbgapi\_wave\_stop\_reason\_t

A bit mask of the reasons that a wave stopped.

The stop reason of a wave is available using the `AMD_DBGAPI_WAVE_INFO_STOP_REASON` query.

## Enumerator

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_NONE** If none of the bits are set, then [amd\\_dbgapi\\_wave\\_stop](#) stopped the wave.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_BREAKPOINT** The wave stopped due to executing a breakpoint instruction.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_WATCHPOINT** The wave stopped due to triggering a data watch point. The [AMD\\_DBGAPI\\_WAVE\\_INFO\\_WATCHPOINTS](#) query can be used to determine which watchpoint(s) were triggered.

The program counter may not be positioned at the instruction that caused the watchpoint(s) to be triggered as the AMD GPU can continue executing instructions after initiating a memory operation. If the architecture supports it, the [amd\\_dbgapi\\_set\\_memory\\_precision](#) can be used to control the precision, but may significantly reduce performance.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_SINGLE\_STEP** The wave stopped due to completing an instruction single-step.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_QUEUE\_ERROR** The wave belongs to a queue that is in the error state. This is set in both waves that were stopped due to a queue error, as well as waves that were already stopped when the queue went into the queue error state.

A wave that includes this stop reason cannot be resumed using [amd\\_dbgapi\\_wave\\_resume](#). The wave's queue will be in the queue error state.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INPUT\_DENORMAL** The wave stopped due to triggering an enabled floating point input denormal exception.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_DIVIDE\_BY\_0** The wave stopped due to triggering an enabled floating point divide by zero exception.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_OVERFLOW** The wave stopped due to triggering an enabled floating point overflow exception.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_UNDERFLOW** The wave stopped due to triggering an enabled floating point underflow exception.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INEXACT** The wave stopped due to triggering an enabled floating point inexact exception.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INVALID\_OPERATION** The wave stopped due to triggering an enabled floating point invalid operation exception.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_INT\_DIVIDE\_BY\_0** The wave stopped due to triggering an enabled integer divide by zero exception.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_DEBUG\_TRAP** The wave stopped due to executing a debug trap instruction. The program counter is left positioned after the trap instruction. The wave can be resumed using [amd\\_dbgapi\\_wave\\_resume](#).

The debug trap instruction can be generated using the `llvm.debugtrap` compiler intrinsic. See [User Guide for AMDGPU Backend - Code Conventions - AMDHSA - Trap Handler ABI](#).

A debug trap can be used to explicitly insert stop points in a program to help debugging. They behave as no operations if a debugger is not connected and stop the wave if executed with the debugger attached.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ASSERT\_TRAP** The wave stopped due to executing an assert trap instruction. The program counter is left positioned at the assert trap instruction.

The trap instruction can be generated using the `llvm.trap` compiler intrinsic. See [User Guide for AMDGPU Backend - Code Conventions - AMDHSA - Trap Handler ABI](#).

An assert trap can be used to abort the execution of the dispatches executing on a queue.

A wave that includes this stop reason cannot be resumed using [amd\\_dbgapi\\_wave\\_resume](#). The wave's queue will enter the queue error state and include the [AMD\\_DBGAPI\\_QUEUE\\_ERROR\\_REASON\\_ASSERT\\_TRAP](#) queue error reason.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_TRAP** The wave stopped due to executing a trap instruction other than the [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_DEBUG\\_TRAP](#) or [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_ASSERT\\_TRAP](#) trap instruction. The program counter is left positioned at the trap instruction.

A wave that includes this stop reason cannot be resumed using [amd\\_dbgapi\\_wave\\_resume](#). The wave's queue will enter the queue error state and include the [AMD\\_DBGAPI\\_QUEUE\\_ERROR\\_REASON\\_WAVE\\_ERROR](#) queue error reason.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_MEMORY\_VIOLATION** The wave stopped due to triggering a memory violation. The program counter may not be positioned at the instruction that caused the memory violation as the AMD GPU can continue executing instructions after initiating a memory operation. If the architecture supports it, the [amd\\_dbgapi\\_set\\_memory\\_precision](#) can be used to control the precision, but may significantly reduce performance.

A wave that includes this stop reason cannot be resumed using [amd\\_dbgapi\\_wave\\_resume](#). The wave's queue will enter the queue error state and include the [AMD\\_DBGAPI\\_QUEUE\\_ERROR\\_REASON\\_MEMORY\\_VIOLATION](#) queue error reason.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ILLEGAL\_INSTRUCTION** The wave stopped due to executing an illegal instruction. The program counter is left positioned at the illegal instruction.

A wave that includes this stop reason cannot be resumed using [amd\\_dbgapi\\_wave\\_resume](#). The wave's queue will enter the queue error state and include the [AMD\\_DBGAPI\\_QUEUE\\_ERROR\\_REASON\\_WAVE\\_ERROR](#) queue error reason.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ECC\_ERROR** The wave stopped due to detecting an unrecoverable ECC error. The program counter may not be positioned at the instruction that caused the memory violation as the AMD GPU can continue executing instructions after initiating a memory operation. If the architecture supports it, the [amd\\_dbgapi\\_set\\_memory\\_precision](#) can be used to control the precision, but may significantly reduce performance.

A wave that includes this stop reason cannot be resumed using [amd\\_dbgapi\\_wave\\_resume](#). The wave's queue will enter the queue error state and include the [AMD\\_DBGAPI\\_QUEUE\\_ERROR\\_REASON\\_WAVE\\_ERROR](#) queue error reason.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FATAL\_HALT** The wave stopped after causing a hardware fatal halt. A wave that includes this stop reason cannot be resumed using [amd\\_dbgapi\\_wave\\_resume](#). The wave's queue will enter the queue error state and include the [AMD\\_DBGAPI\\_QUEUE\\_ERROR\\_REASON\\_WAVE\\_ERROR](#) queue error reason.

**AMD\_DBGAPI\_WAVE\_STOP\_REASON\_XNACK\_ERROR** The wave stopped with an XNACK error. A wave that includes this stop reason cannot be resumed using [amd\\_dbgapi\\_wave\\_resume](#). The wave's queue will enter the queue error state and include the [AMD\\_DBGAPI\\_QUEUE\\_ERROR\\_REASON\\_WAVE\\_ERROR](#) queue error reason.

## 2.11.4 Function Documentation

2.11.4.1 **amd\_dbgapi\_status\_t** **AMD\_DBGAPI** [amd\\_dbgapi\\_wave\\_get\\_info](#) ( [amd\\_dbgapi\\_process\\_id\\_t](#) *process\_id*, [amd\\_dbgapi\\_wave\\_id\\_t](#) *wave\_id*, [amd\\_dbgapi\\_wave\\_info\\_t](#) *query*, [size\\_t](#) *value\_size*, void \* *value* )

Query information about a wave.

[amd\\_dbgapi\\_wave\\_info\\_t](#) specifies the queries supported and the type returned using the *value* argument.

Parameters

in	<i>process_id</i>	The process to which the queue belongs.
----	-------------------	---

in	<i>wave_id</i>	The handle of the wave being queried.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <i>value</i> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

**Return values**

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>value</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<i>wave_id</i> is invalid. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>value</i> is NULL or <i>query</i> is invalid. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<i>value_size</i> does not match the size of the result. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <i>value</i> returns NULL. <i>value</i> is unaltered.

**2.11.4.2** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_wave_list ( amd_dbgapi_process_id_t process_id, size_t * wave_count, amd_dbgapi_wave_id_t ** waves, amd_dbgapi_changed_t * changed )`

Return the list of existing waves for a process.

The order of the wave handles in the list is unspecified and can vary between calls.

**Parameters**

in	<i>process_id</i>	The process for which the wave list is requested.
out	<i>wave_count</i>	The number of waves executing in the process.
out	<i>waves</i>	If <i>changed</i> is not NULL and the wave list has not changed since the last call to <a href="#">amd_dbgapi_wave_list</a> then return NULL. Otherwise, return a pointer to an array of <a href="#">amd_dbgapi_wave_id_t</a> with <i>wave_count</i> elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.
in, out	<i>changed</i>	If NULL then left unaltered. If non-NULL, set to <a href="#">AMD_DBGAPI_CHANGED_NO</a> if the list of waves is the same as when <a href="#">amd_dbgapi_wave_list</a> was last called, otherwise set to <a href="#">AMD_DBGAPI_CHANGED_YES</a> .

**Return values**

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>changed</i> , <i>wave_count</i> , and <i>waves</i> .
---	---

<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>changed</code> , <code>wave_count</code> , and <code>waves</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>wave_count</code> , <code>waves</code> , and <code>changed</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. <code>wave_count</code> , <code>waves</code> , and <code>unchanged</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>wave_count</code> or <code>waves</code> are NULL, or <code>changed</code> is invalid. <code>wave_count</code> , <code>waves</code> , and <code>changed</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <code>waves</code> returns NULL. <code>wave_count</code> , <code>waves</code> , and <code>changed</code> are unaltered.

#### 2.11.4.3 `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_wave_resume ( amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_resume_mode_t resume_mode )`

Resume execution of a stopped wave.

The wave can be resumed normally in which case it will be in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_RUN](#) state and be available for the hardware to execute instructions. Just because it is in the run state does not mean the hardware will start executing instructions immediately as that depends on the AMD GPU hardware scheduler.

If while in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_RUN](#) state, the wave encounters something that stops its execution, or [amd\\_dbgapi\\_wave\\_stop](#) is used to stop the wave execution, then a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event will be created.

If while in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_RUN](#) state the wave terminates, no event is created.

The wave can be resumed in single step mode in which case it will be in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_SINGLE\\_STEP](#) state. It is available for the hardware to execute one instruction. After completing execution of a regular instruction, a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event will be created that indicates the wave has stopped. The stop reason of the wave will include [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_SINGLE\\_STEP](#). After completing execution of a wave termination instruction, a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_COMMAND\\_TERMINATED](#) event will be created that indicates that the wave has terminated. On some architectures, a single step that completes with the wave positioned at a wave termination instruction may also report the [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_COMMAND\\_TERMINATED](#) event.

Resuming a wave in single step mode does not necessarily cause it to execute any instructions as it is up to the AMD GPU hardware scheduler to decide what waves to execute. For example, the AMD GPU hardware scheduler may not execute any instructions of a wave until other waves have terminated. If the client has stopped other waves this can prevent a wave from ever performing a single step. The client should handle this gracefully and not rely on a single step request always resulting in a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event. If necessary, the client should respond to the stop events of other waves to allow them to make forward progress, and handle the single step stop request when it finally arrives. If necessary, the client can cancel the single step request by using [amd\\_dbgapi\\_wave\\_stop](#) and allow the user to attempt it again later when other waves have terminated.

It is an error to resume a wave that has terminated. The wave handle will be reported as invalid. It is up to the client to use [amd\\_dbgapi\\_wave\\_list](#) to determine what waves have been created and terminated. No event is reported when a wave is created or terminates.

It is an error to request a wave to resume that is not in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_STOP](#) state, or is in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_STOP](#) state but the [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event that put it in the stop state has not yet been completed using the [amd\\_dbgapi\\_event\\_processed](#) operation. Therefore, it is not allowed to execute multiple resume requests as all but the first one will give an error.

It also means it is an error to resume a wave that has already stopped, but whose [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WA-](#)

[VE\\_STOP](#) event has not yet been returned by [amd\\_dbgapi\\_next\\_pending\\_event](#), since the wave is still in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_RUN](#) state. The [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) must be processed first.

Since a resume request can only be sent to a wave that has stopped, there is no issue of the wave terminating while making the request. However, the wave may terminate after being resumed. Except for single stepping the wave termination instruction described above, no event is reported when the wave terminates.

Sending a resume request to a wave that includes a stop reason that cannot be resumed will report an error. See [amd\\_dbgapi\\_wave\\_stop\\_reason\\_t](#).

#### Parameters

in	<i>process_id</i>	The process to which the wave belongs.
in	<i>wave_id</i>	The wave being requested to resume.
in	<i>resume_mode</i>	If <a href="#">AMD_DBGAPI_RESUME_MODE_NORMAL</a> , then resume normal execution of the wave. If <a href="#">AMD_DBGAPI_RESUME_MODE_SINGLE_STEP</a> , then resume the wave in single step mode.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the wave will either terminate or be stopped. In either case a <a href="#">AMD_DBGAPI_EVENT_KIND_WAVE_STOP</a> event will be reported.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and no wave is resumed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. No wave is resumed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<i>wave_id</i> is invalid. No wave is resumed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>resume_mode</i> is invalid. No wave is resumed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED</a>	<i>wave_id</i> is not stopped. The wave remains running.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_RESUMABLE</a>	<i>wave_id</i> is stopped with a reason that includes one that cannot be resumed.

2.11.4.4 **amd\_dbgapi\_status\_t** **AMD\_DBGAPI** **amd\_dbgapi\_wave\_stop** ( **amd\_dbgapi\_process\_id\_t** *process\_id*, **amd\_dbgapi\_wave\_id\_t** *wave\_id* )

Request a wave to stop executing.

The wave may or may not immediately stop. If the wave does not immediately stop, the stop request is termed outstanding until the wave does stop or the wave terminates before stopping. When the wave does stop it will create a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event. If the wave terminates before stopping it will create a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_COMMAND\\_TERMINATED](#) event.

It is an error to request a wave to stop that has terminated. The wave handle will be reported as invalid. It is up to the client to use [amd\\_dbgapi\\_wave\\_list](#) to determine what waves have been created and terminated. No event is reported when a wave is created or terminates.

It is an error to request a wave to stop that is already in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_STOP](#) state.

It is an error to request a wave to stop for which there is an outstanding [amd\\_dbgapi\\_wave\\_stop](#) request.

Sending a stop request to a wave that has already stopped, but whose [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event has not yet been returned by [amd\\_dbgapi\\_next\\_pending\\_event](#), is allowed since the wave is still in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_RUN](#) state. In this case the wave is not affected and the already existing [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) will notify the client that the stop request has completed. The client must be prepared that a wave may stop for other reasons in response to a stop request. It can use the [AMD\\_DBGAPI\\_WAVE\\_INFO\\_STOP\\_REASON](#) query to determine if there are other reason(s). See [AMD\\_DBGAPI\\_WAVE\\_STATE\\_STOP](#) for more information.

Sending a stop request to a wave that is in the [AMD\\_DBGAPI\\_WAVE\\_STATE\\_SINGLE\\_STEP](#) state will attempt to stop the wave and either report a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) or [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_COMMAND\\_TERMINATED](#) event. If the wave did stop, the setting of the [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_SINGLE\\_STEP](#) stop reason will indicate whether the wave completed the single step. If the single step does complete, but terminates the wave, then [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_COMMAND\\_TERMINATED](#) will be reported.

Sending a stop request to a wave that is present at the time of the request, and does stop, will result in a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event.

Sending a stop request to a wave that is present at the time of the request, but terminates before completing the stop request, will result in a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_COMMAND\\_TERMINATED](#) event.

#### Parameters

in	<i>process_id</i>	The process to which the wave belongs.
in	<i>wave_id</i>	The wave being requested to stop.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the wave will either report a <a href="#">AMD_DBGAPI_EVENT_KIND_WAVE_STOP</a> or <a href="#">AMD_DBGAPI_EVENT_KIND_WAVE_COMMAND_TERMINATED</a> event.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and no wave is stopped.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. No wave is stopped.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<i>wave_id</i> is invalid. No wave is stopped.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_STOPPED</a>	The wave is already stopped. The wave remains stopped.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_OUTSTANDING_STOP</a>	The wave already has an outstanding stop request. This stop request is ignored and the previous stop request continues to stop the wave.



## 2.12 Displaced Stepping

Operations related to AMD GPU breakpoint displaced stepping.

### Data Structures

- struct `amd_dbgapi_displaced_stepping_id_t`  
*Opaque displaced stepping handle.*

### Macros

- `#define AMD_DBGAPI_DISPLACED_STEPPING_NONE (amd_dbgapi_displaced_stepping_id_t{ 0 })`  
*The NULL displaced stepping handle.*

### Functions

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_displaced_stepping_start (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, const void *saved_instruction_bytes, amd_dbgapi_displaced_stepping_id_t *displaced_stepping)`  
*Create a displaced stepping buffer.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_displaced_stepping_complete (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_displaced_stepping_id_t displaced_stepping)`  
*Complete a displaced stepping buffer for a wave.*

#### 2.12.1 Detailed Description

Operations related to AMD GPU breakpoint displaced stepping. The AMD GPU does not support halting of wave creation since there is fixed function hardware that performs wave creation of dispatches. It is therefore not possible to step over a breakpoint by halting all waves, assuming that will also prevent new wave creation, removing the breakpoint instruction by replacing it with the original instruction bytes, single stepping it, and finally replacing the breakpoint instruction. Doing so may result in newly created waves missing the removed breakpoint.

Instead the library supports displaced stepping buffers. These allow the instruction that is overwritten by a breakpoint instruction to be copied to a buffer and single stepped in the buffer. This does not require the breakpoint instruction to be removed and so breakpoints cannot be missed even if other waves are executing or are created. It also avoids having to stop all other waves.

When an instruction is copied into a displaced stepping buffer, it may be necessary to modify the instruction, or its register inputs to account for the fact that it is executing at a different address. Similarly, after single stepping it, registers and program counter may need adjusting. It may also be possible to simply know the effect of an instruction and avoid single stepping it at all and simply update the wave state directly. For example, branches can be trivial to emulate this way.

The operations in this section allow displaced stepping buffers to be allocated and used. They will take care of all the architecture specific details described above.

The number of displaced stepping buffers supported by the library is not fixed, but there is always at least one. It may be possible for the library to share the same displaced stepping buffer with multiple waves. For example, if the waves are at the same breakpoint. The library will determine when this is possible, but the client should not rely on this. Some waves at the same breakpoint may be able to share while others may not. The client must handle the case when there are no more displaced stepping buffers available. The client may be able to maximize the number of waves it can single



step at once by requesting displaced stepping buffers for all waves at the same breakpoint. Just because there is no displaced stepping buffer for one wave, does not mean another wave cannot be assigned to a displaced stepping buffer through sharing.

If allocating a displaced stepping buffer indicates that the wave has already been single stepped over the breakpoint, the client can simply resume the wave normally.

If allocating a displaced stepping buffer indicates no more are available, the client must complete using the previously allocated buffers and release them before trying again.

If allocating a displaced stepping buffer is successful, then the client must resume the wave in single step mode. When the single step has completed, the buffer can be released, and the wave resumed normally.

If the wave does not complete the single step, then the wave can be stopped, and the buffer released. If the single step did not complete then this will leave the wave still at the breakpoint, and the client can retry stepping over the breakpoint later.

See Also

[amd\\_dbgapi\\_wave\\_resume](#), [amd\\_dbgapi\\_wave\\_stop](#)

## 2.12.2 Macro Definition Documentation

### 2.12.2.1 `#define AMD_DBGAPI_DISPLACED_STEPPING_NONE (amd_dbgapi_displaced_stepping_id_t{ 0 })`

The NULL displaced stepping handle.

## 2.12.3 Function Documentation

### 2.12.3.1 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_displaced_stepping_complete ( amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_displaced_stepping_id_t displaced_stepping )`

Complete a displaced stepping buffer for a wave.

The wave must be stopped and have been set to use the stepping buffer by using [amd\\_dbgapi\\_displaced\\_stepping\\_start](#).

If the wave single step has not completed the wave state is reset to what it was before [amd\\_dbgapi\\_displaced\\_stepping\\_start](#). The wave is left stopped and the client can retry stepping over the breakpoint again later.

If the single step has completed, then the wave state is updated to be after the instruction at which the breakpoint instruction is placed. The wave program counter and other registers may be changed so the client should flush any cached register values. The wave is left stopped and can be resumed normally by the client.

If the wave is the last one using the displaced stepping buffer, the buffer is freed and the handle invalidated.

Parameters

in	<i>process_id</i>	The process to which the wave belongs.
in	<i>wave_id</i>	The wave using the displaced stepping buffer.
in	<i>displaced_stepping</i>	The displaced stepping buffer to complete.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully. The displaced stepping buffer is completed, and the wave is either stepped over the breakpoint, or still at the breakpoint.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized, no displaced stepping buffer is completed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized, no displaced stepping buffer completed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. No displaced stepping buffer is completed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<code>wave_id</code> is invalid. No displaced stepping buffer is completed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_DISPLACED_STEPPING_ID</a>	<code>displaced_stepping</code> is invalid or not in use by <code>wave_id</code> . No displaced stepping buffer is completed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED</a>	<code>wave_id</code> is not stopped. No displaced stepping buffer is completed.

2.12.3.2 `amd_dbgapi_status_t` **AMD\_DBGAPI** `amd_dbgapi_displaced_stepping_start ( amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, const void * saved_instruction_bytes, amd_dbgapi_displaced_stepping_id_t * displaced_stepping )`

Create a displaced stepping buffer.

The wave must be stopped.

Displaced stepping buffers are intended to be used to step over breakpoints. In that case, the wave will be stopped with a program counter set to a breakpoint instruction that was placed by the client overwriting all or part of the original instruction where the breakpoint was placed. The client must provide the overwritten bytes of the original instruction.

If [AMD\\_DBGAPI\\_DISPLACED\\_STEPPING\\_NONE](#) is returned successfully it indicates the wave has been single stepped over the breakpoint. The wave is still stopped and is available to be resumed normally.

If a displaced stepping handle is returned successfully, the wave is still stopped. The wave program counter and other registers may be changed so the client should flush any cached register values. The client should resume the wave in single step mode using [amd\\_dbgapi\\_wave\\_resume](#). Once the single step is complete as indicated by the [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event with a stop reason that includes [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_SINGLE\\_STEP](#), the client should use [amd\\_dbgapi\\_displaced\\_stepping\\_complete](#) to release the displaced stepping buffer. The wave can then be resumed normally using [amd\\_dbgapi\\_wave\\_resume](#).

If the single step is cancelled by stopping the wave, the client must determine if the wave completed the single step to determine if the wave can be resumed or must retry the displaced stepping later. See [amd\\_dbgapi\\_wave\\_stop](#).

## Parameters

in	<i>process_id</i>	The process to which the wave belongs.
in	<i>wave_id</i>	The wave to create a displaced stepping buffer.
in	<i>saved_instruction_bytes</i>	The original instruction bytes that the breakpoint instruction replaced. The number of bytes must be <a href="#">AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_SIZE</a> .

out	<i>displaced_ - stepping</i>	The displace stepping handle, or <a href="#">AMD_DBGAPI_DISPLACED_STEPPING_NONE</a> .
-----	----------------------------------	---

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and <code>displaced_stepping</code> is set to <a href="#">AMD_DBGAPI_DISPLACED_STEPPING_NONE</a> or to a valid displaced stepping handle.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized, no displaced stepping buffer is allocated and <code>displaced_stepping</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized, no displaced stepping buffer is allocated and <code>displaced_stepping</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. No displaced stepping buffer is allocated and <code>displaced_stepping</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<code>wave_id</code> is invalid. No displaced stepping buffer is allocated and <code>displaced_stepping</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED</a>	<code>wave_id</code> is not stopped. No displaced stepping buffer is allocated and <code>displaced_stepping</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_DISPLACED_STEPPING_BUFFER_UNAVAILABLE</a>	No more displaced stepping buffers are available that are suitable for use by <code>wave_id</code> . No displaced stepping buffer is allocated and <code>displaced_stepping</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>original_instruction</code> or <code>displaced_stepping</code> are NULL. No displaced stepping buffer is allocated and <code>displaced_stepping</code> is unaltered.

## 2.13 Watchpoints

Operations related to AMD GPU hardware data watchpoints.

### Macros

- `#define AMD_DBGAPI_WATCHPOINT_NONE (amd_dbgapi_watchpoint_id_t (-1))`  
*The NULL watchpoint handle.*

### Typedefs

- `typedef uint32_t amd_dbgapi_watchpoint_id_t`  
*A hardware data watchpoint handle.*

### Enumerations

- `enum amd_dbgapi_watchpoint_share_kind_t { AMD_DBGAPI_WATCHPOINT_SHARE_KIND_UNSUPPORTED = 0, AMD_DBGAPI_WATCHPOINT_SHARE_KIND_UNSHARED = 1, AMD_DBGAPI_WATCHPOINT_SHARE_KIND_SHARED = 2 }`  
*The way watchpoints are shared between processes.*
- `enum amd_dbgapi_watchpoint_kind_t { AMD_DBGAPI_WATCHPOINT_KIND_LOAD = 1, AMD_DBGAPI_WATCHPOINT_KIND_STORE_AND_RMW = 2, AMD_DBGAPI_WATCHPOINT_KIND_RMW = 3, AMD_DBGAPI_WATCHPOINT_KIND_ALL = 4 }`  
*Watchpoint memory access kinds.*

### Functions

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_set_watchpoint (amd_dbgapi_process_id_t process_id, amd_dbgapi_agent_id_t agent_id, amd_dbgapi_global_address_t address, amd_dbgapi_size_t size, amd_dbgapi_watchpoint_kind_t kind, amd_dbgapi_watchpoint_id_t *watchpoint_id, amd_dbgapi_global_address_t *watchpoint_address, amd_dbgapi_size_t *watchpoint_size)`  
*Set a hardware data watchpoint.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_remove_watchpoint (amd_dbgapi_process_id_t process_id, amd_dbgapi_agent_id_t agent_id, amd_dbgapi_watchpoint_id_t watchpoint_id)`  
*Remove a hardware data watchpoint previously set by [amd\\_dbgapi\\_set\\_watchpoint](#).*

#### 2.13.1 Detailed Description

Operations related to AMD GPU hardware data watchpoints. A data watchpoint is a hardware supported mechanism to generate wave stop events when a wave accesses memory in a certain way in a certain address range.

The granularity of base address and address range is architecture specific.

The number of watchpoints supported by an architecture is available using the [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_WATCHPOINT\\_COUNT](#) query and may be 0. The [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_WATCHPOINT\\_SHARE](#) query can be used to determine if watchpoints are shared between processes using the same agent.

When a wave stops due to a data watch point the stop reason will include [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_WATCHPOINT](#). The set of watchpoints triggered can be queried using [AMD\\_DBGAPI\\_WAVE\\_INFO\\_WATCHPOINTS](#).

### 2.13.2 Macro Definition Documentation

#### 2.13.2.1 `#define AMD_DBGAPI_WATCHPOINT_NONE (amd_dbgapi_watchpoint_id_t (-1))`

The NULL watchpoint handle.

### 2.13.3 Typedef Documentation

#### 2.13.3.1 `typedef uint32_t amd_dbgapi_watchpoint_id_t`

A hardware data watchpoint handle.

Hardware data watchpoints are numbered from 0 to [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_WATCHPOINT\\_COUNT](#) minus 1.

Only unique for a single agent of a single process.

### 2.13.4 Enumeration Type Documentation

#### 2.13.4.1 `enum amd_dbgapi_watchpoint_kind_t`

Watchpoint memory access kinds.

The watchpoint is triggered only when the memory instruction is of the specified kind.

Enumerator

**`AMD_DBGAPI_WATCHPOINT_KIND_LOAD`** Read access by load instructions.

**`AMD_DBGAPI_WATCHPOINT_KIND_STORE_AND_RMW`** Write access by store instructions or read-modify-write access by atomic instructions.

**`AMD_DBGAPI_WATCHPOINT_KIND_RMW`** Read-modify-write access by atomic instructions.

**`AMD_DBGAPI_WATCHPOINT_KIND_ALL`** Read, write, or read-modify-write access by load, store, or atomic instructions.

#### 2.13.4.2 `enum amd_dbgapi_watchpoint_share_kind_t`

The way watchpoints are shared between processes.

The [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_WATCHPOINT\\_SHARE](#) query can be used to determine the watchpoint sharing for an architecture.

Enumerator

**`AMD_DBGAPI_WATCHPOINT_SHARE_KIND_UNSUPPORTED`** Watchpoints are not supported.

**`AMD_DBGAPI_WATCHPOINT_SHARE_KIND_UNSHARED`** Watchpoints are not shared. Every process using an agent can use all [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_WATCHPOINT\\_COUNT](#) watchpoints.

**`AMD_DBGAPI_WATCHPOINT_SHARE_KIND_SHARED`** Watchpoints are shared. The number of watchpoints available to a process may be reduced if watchpoints are used by another process.

### 2.13.5 Function Documentation

2.13.5.1 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_remove_watchpoint ( amd_dbgapi_process_id_t process_id, amd_dbgapi_agent_id_t agent_id, amd_dbgapi_watchpoint_id_t watchpoint_id )`

Remove a hardware data watchpoint previously set by [amd\\_dbgapi\\_set\\_watchpoint](#).

## Parameters

in	<i>process_id</i>	The process to which the agent belongs.
in	<i>agent_id</i>	Specify the agent that owns the watchpoint.
in	<i>watchpoint_id</i>	The watchpoint to remove.

## Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	The function has been executed successfully and the watchpoint has been removed.
<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and no watchpoint is removed.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</i></a>	<i>process_id</i> is invalid. No watchpoint is removed.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_AGENT_ID</i></a>	<i>agent_id</i> is invalid. No watchpoint is removed.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_WATCHPOINT_ID</i></a>	<i>watchpoint_id</i> is invalid. No watchpoint is removed.

2.13.5.2 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_set_watchpoint ( amd_dbgapi_process_id_t process_id, amd_dbgapi_agent_id_t agent_id, amd_dbgapi_global_address_t address, amd_dbgapi_size_t size, amd_dbgapi_watchpoint_kind_t kind, amd_dbgapi_watchpoint_id_t * watchpoint_id, amd_dbgapi_global_address_t * watchpoint_address, amd_dbgapi_size_t * watchpoint_size )`

Set a hardware data watchpoint.

The AMD GPU has limitations on the base address and size of hardware data watchpoints that can be set, and the limitations may vary by architecture. A watchpoint is created with the smallest range that covers the requested range specified by *address* and *size*. The range of the created watchpoint is returned in *watchpoint\_address* and *watchpoint\_size*.

When a watchpoint is triggered, the client is responsible for determining if the access was to the requested range. For example, for writes the client can compare the original value with the current value to determine if it changed.

Each agent has its own set of watchpoints. Only waves executing on the agent will trigger the watchpoints set on that agent.

## Parameters

in	<i>process_id</i>	The process to which the agent belongs.
in	<i>agent_id</i>	Specify the agent to set the watchpoint.
in	<i>address</i>	The base address of memory area to set a watchpoint.
in	<i>size</i>	The number of bytes that the watchpoint should cover.
in	<i>kind</i>	The kind of memory access that should trigger the watchpoint.
out	<i>watchpoint_id</i>	The watchpoint created.
out	<i>watchpoint - address</i>	The base address of the created watchpoint.

out	<i>watchpoint_size</i>	The byte size of the created watchpoint.
-----	------------------------	--

## Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	The function has been executed successfully and the watchpoint has been created with handle <i>watchpoint_id</i> that covers the range specified by <i>watchpoint_address</i> and <i>watchpoint_size</i> .
<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized and <i>watchpoint_id</i> , <i>watchpoint_address</i> , and <i>watchpoint_size</i> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and <i>watchpoint_id</i> , <i>watchpoint_address</i> , and <i>watchpoint_size</i> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</i></a>	<i>process_id</i> is invalid. No watchpoint is set and <i>watchpoint_id</i> , <i>watchpoint_address</i> , and <i>watchpoint_size</i> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_AGENT_ID</i></a>	<i>agent_id</i> is invalid. No watchpoint is set and <i>watchpoint_id</i> , <i>watchpoint_address</i> , and <i>watchpoint_size</i> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NO_WATCHPOINT_AVAILABLE</i></a>	No more watchpoints are available. No watchpoint is set and <i>watchpoint_id</i> , <i>watchpoint_address</i> , and <i>watchpoint_size</i> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_SUPPORTED</i></a>	Watchpoints are not supported for the architecture of the agent. No watchpoint is set and <i>watchpoint_id</i> , <i>watchpoint_address</i> , and <i>watchpoint_size</i> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</i></a>	<i>kind</i> is invalid; or <i>watchpoint_id</i> , <i>watchpoint_address</i> , or <i>watchpoint_size</i> are NULL. No watchpoint is set and <i>watchpoint_id</i> , <i>watchpoint_address</i> , and <i>watchpoint_size</i> are unaltered.



## 2.14 Registers

Operations related to AMD GPU register access.

### Data Structures

- struct `amd_dbgapi_register_class_id_t`  
*Opaque register class handle.*
- struct `amd_dbgapi_register_id_t`  
*Opaque register handle.*

### Macros

- #define `AMD_DBGAPI_REGISTER_CLASS_NONE` (`amd_dbgapi_register_class_id_t{ 0 }`)  
*The NULL register class handle.*
- #define `AMD_DBGAPI_REGISTER_NONE` (`amd_dbgapi_register_id_t{ 0 }`)  
*The NULL register handle.*

### Enumerations

- enum `amd_dbgapi_register_class_info_t` { `AMD_DBGAPI_REGISTER_CLASS_INFO_NAME` = 1 }  
*Register class queries that are supported by `amd_dbgapi_architecture_register_class_get_info`.*
- enum `amd_dbgapi_register_info_t` { `AMD_DBGAPI_REGISTER_INFO_NAME` = 1, `AMD_DBGAPI_REGISTER_INFO_SIZE` = 2, `AMD_DBGAPI_REGISTER_INFO_TYPE` = 3 }  
*Register queries that are supported by `amd_dbgapi_architecture_register_get_info` and `amd_dbgapi_wave_register_get_info`.*
- enum `amd_dbgapi_register_class_state_t` { `AMD_DBGAPI_REGISTER_CLASS_STATE_NOT_MEMBER` = 0, `AMD_DBGAPI_REGISTER_CLASS_STATE_MEMBER` = 1 }  
*Indication of whether a register is a member of a register class.*

### Functions

- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_architecture_register_class_get_info` (`amd_dbgapi_architecture_id_t` `architecture_id`, `amd_dbgapi_register_class_id_t` `register_class_id`, `amd_dbgapi_register_class_info_t` `query`, `size_t` `value_size`, `void *``value`)  
*Query information about a register class of an architecture.*
- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_architecture_register_class_list` (`amd_dbgapi_architecture_id_t` `architecture_id`, `size_t` `*register_class_count`, `amd_dbgapi_register_class_id_t` `**register_classes`)  
*Report the list of register classes supported by the architecture.*
- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_architecture_register_get_info` (`amd_dbgapi_architecture_id_t` `architecture_id`, `amd_dbgapi_register_id_t` `register_id`, `amd_dbgapi_register_info_t` `query`, `size_t` `value_size`, `void *``value`)  
*Query information about a register of an architecture.*
- `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_wave_register_get_info` (`amd_dbgapi_process_id_t` `process_id`, `amd_dbgapi_wave_id_t` `wave_id`, `amd_dbgapi_register_id_t` `register_id`, `amd_dbgapi_register_info_t` `query`, `size_t` `value_size`, `void *``value`)  
*Query information about a register of a wave.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_architecture\\_register\\_list](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [size\\_t](#) \*register\_count, [amd\\_dbgapi\\_register\\_id\\_t](#) \*\*registers)

*Report the list of registers supported by the architecture.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_wave\\_register\\_list](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [size\\_t](#) \*register\_count, [amd\\_dbgapi\\_register\\_id\\_t](#) \*\*registers)

*Report the list of registers supported by a wave.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_dwarf\\_register\\_to\\_register](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [uint64\\_t](#) dwarf\_register, [amd\\_dbgapi\\_register\\_id\\_t](#) \*register\_id)

*Return a register handle from an AMD GPU DWARF register number.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_register\\_is\\_in\\_register\\_class](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_register\\_class\\_id\\_t](#) register\_class\_id, [amd\\_dbgapi\\_register\\_class\\_state\\_t](#) \*register\_class\_state)

*Determine if a register is a member of a register class.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_read\\_register](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_size\\_t](#) offset, [amd\\_dbgapi\\_size\\_t](#) value\_size, [void](#) \*value)

*Read a register.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_write\\_register](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_size\\_t](#) offset, [amd\\_dbgapi\\_size\\_t](#) value\_size, [const void](#) \*value)

*Write a register.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_prefetch\\_register](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_size\\_t](#) register\_count)

*Prefetch register values.*

## 2.14.1 Detailed Description

Operations related to AMD GPU register access.

## 2.14.2 Macro Definition Documentation

### 2.14.2.1 `#define AMD_DBGAPI_REGISTER_CLASS_NONE (amd_dbgapi_register_class_id_t{ 0 })`

The NULL register class handle.

### 2.14.2.2 `#define AMD_DBGAPI_REGISTER_NONE (amd_dbgapi_register_id_t{ 0 })`

The NULL register handle.

## 2.14.3 Enumeration Type Documentation

### 2.14.3.1 `enum amd_dbgapi_register_class_info_t`

Register class queries that are supported by [amd\\_dbgapi\\_architecture\\_register\\_class\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_architecture\\_register\\_class\\_get\\_info](#).

## Enumerator

**AMD\_DBGAPI\_REGISTER\_CLASS\_INFO\_NAME** Return the register class name. The type of this attribute is a pointer to a NUL terminated `char`. It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

2.14.3.2 `enum amd_dbgapi_register_class_state_t`

Indication of whether a register is a member of a register class.

## Enumerator

**AMD\_DBGAPI\_REGISTER\_CLASS\_STATE\_NOT\_MEMBER** The register is not a member of the register class.

**AMD\_DBGAPI\_REGISTER\_CLASS\_STATE\_MEMBER** The register is a member of the register class.

2.14.3.3 `enum amd_dbgapi_register_info_t`

Register queries that are supported by [amd\\_dbgapi\\_architecture\\_register\\_get\\_info](#) and [amd\\_dbgapi\\_wave\\_register\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_architecture\\_register\\_get\\_info](#) and [amd\\_dbgapi\\_wave\\_register\\_get\\_info](#).

## Enumerator

**AMD\_DBGAPI\_REGISTER\_INFO\_NAME** Return the register name. The type of this attribute is a pointer to a NUL terminated `char`. It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

**AMD\_DBGAPI\_REGISTER\_INFO\_SIZE** Return the size of the register in bytes. The size of a register may vary depending on the lane count of the wave which can be obtained by the [AMD\\_DBGAPI\\_WAVE\\_INFO\\_LANE\\_COUNT](#) query. For example, the execution mask register, condition code register, and all vector registers vary by the lane count of the wave. Not supported for the [amd\\_dbgapi\\_architecture\\_register\\_get\\_info](#). The type of this attribute is [amd\\_dbgapi\\_size\\_t](#).

**AMD\_DBGAPI\_REGISTER\_INFO\_TYPE** Return the register type as a C style type string. This can be used as the default type to use when displaying values of the register. The type string syntax is defined by the following BNF syntax:

```
type ::= integer_type | float_type | array_type | function_type
integer_type ::= "uint32" | "uint64"
float_type ::= "float" | "double"
array_type ::= ( integer_type | float_type ) "[" integer "]"
function_type ::= "void(void)"
integer ::= digit | ( digit integer )
digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The type size matches the size of the register. `uint32` and `float` types are 4 bytes. `uint64` and `double` types are 8 bytes. `void(void)` is the size of a global address.

The type of this attribute is a pointer to a NUL terminated `char`. It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

## 2.14.4 Function Documentation

2.14.4.1 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_architecture\_register\_class\_get\_info (   
**amd\_dbgapi\_architecture\_id\_t** *architecture\_id*, **amd\_dbgapi\_register\_class\_id\_t** *register\_class\_id*,   
**amd\_dbgapi\_register\_class\_info\_t** *query*, **size\_t** *value\_size*, **void \*** *value* )

Query information about a register class of an architecture.

[amd\\_dbgapi\\_register\\_class\\_info\\_t](#) specifies the queries supported and the type returned using the `value` argument.

#### Parameters

in	<i>architecture_id</i>	The architecture to which the register class belongs.
in	<i>register_class_id</i>	The handle of the register class being queried.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <code>value</code> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <code>value</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<code>architecture_id</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_CLASS_ID</a>	<code>register_class_id</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>value</code> is NULL or <code>query</code> is invalid. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<code>value_size</code> does not match the size of the result. <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <code>value</code> returns NULL. <code>value</code> is unaltered.

2.14.4.2 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_architecture\_register\_class\_list ( **amd\_dbgapi\_architecture\_id\_t** *architecture\_id*, **size\_t \*** *register\_class\_count*, **amd\_dbgapi\_register\_class\_id\_t \*** *register\_classes* )

Report the list of register classes supported by the architecture.

The order of the register handles in the list is stable between calls.

#### Parameters

in	<i>architecture_id</i>	The architecture being queried.
----	------------------------	---------------------------------

out	<i>register_class_count</i>	The number of architecture register classes.
out	<i>register_classes</i>	A pointer to an array of <a href="#">amd_dbgapi_register_class_id_t</a> with <i>register_class_count</i> elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>register_class_count</i> and <i>register_classes</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>register_class_count</i> and <i>register_classes</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>register_class_count</i> and <i>register_classes</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<i>architecture_id</i> is invalid. <i>register_class_count</i> and <i>register_classes</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>register_class_count</i> or <i>register_classes</i> are NULL. <i>register_class_count</i> and <i>register_classes</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <i>register_classes</i> returns NULL. <i>register_class_count</i> and <i>register_classes</i> are unaltered.

2.14.4.3 **amd\_dbgapi\_status\_t** **AMD\_DBGAPI** **amd\_dbgapi\_architecture\_register\_get\_info**( **amd\_dbgapi\_architecture\_id\_t** *architecture\_id*, **amd\_dbgapi\_register\_id\_t** *register\_id*, **amd\_dbgapi\_register\_info\_t** *query*, **size\_t** *value\_size*, **void \****value* )

Query information about a register of an architecture.

[amd\\_dbgapi\\_register\\_info\\_t](#) specifies the queries supported and the type returned using the *value* argument.

## Parameters

in	<i>architecture_id</i>	The architecture to which the register belongs.
in	<i>register_id</i>	The handle of the register being queried.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <i>value</i> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>value</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>value</i> is unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	wave_id is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID</a>	register_id is invalid for architecture_id. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	value is NULL, or query is invalid or not supported for an architecture. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	value_size does not match the size of the result. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate value returns NULL. value is unaltered.

#### 2.14.4.4 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_architecture_register_list ( amd_dbgapi_architecture_id_t architecture_id, size_t * register_count, amd_dbgapi_register_id_t ** registers )`

Report the list of registers supported by the architecture.

This list is all the registers the architecture can support, but a specific wave may not have all these registers. For example, AMD GPU architectures can specify the number of vector and scalar registers when a wave is created. Use the [amd\\_dbgapi\\_wave\\_register\\_list](#) operation to determine the registers supported by a specific wave.

The order of the register handles in the list is stable between calls and registers on the same major class are contiguous in ascending hardware number order.

##### Parameters

in	<i>architecture_id</i>	The architecture being queried.
out	<i>register_count</i>	The number of architecture registers.
out	<i>registers</i>	A pointer to an array of <a href="#">amd_dbgapi_register_id_t</a> with <i>register_count</i> elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.

##### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>register_count</i> and <i>registers</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>register_count</i> and <i>registers</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>register_count</i> and <i>registers</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<i>architecture_id</i> is invalid. <i>register_count</i> and <i>registers</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>register_count</i> or <i>registers</i> are NULL. <i>register_count</i> and <i>registers</i> are unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate registers returns NULL. <code>register_count</code> and <code>registers</code> are unaltered.
---	--

2.14.4.5 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_dwarf\_register\_to\_register ( **amd\_dbgapi\_architecture\_id\_t** *architecture\_id*, **uint64\_t** *dwarf\_register*, **amd\_dbgapi\_register\_id\_t** \* *register\_id* )

Return a register handle from an AMD GPU DWARF register number.

See [User Guide for AMDGPU Backend - Code Object - DWARF - Register Mapping](#).

#### Parameters

in	<i>architecture_id</i>	The architecture of the DWARF register.
in	<i>dwarf_register</i>	The AMD GPU DWARF register number.
out	<i>register_id</i>	The register handle that corresponds to the DWARF register ID.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <code>register_id</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>register_id</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>register_id</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<code>architecture_id</code> is invalid. <code>register_id</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>dwarf_register_id</code> is not valid for the architecture or <code>register_id</code> is NULL. <code>register_id</code> is unaltered.

2.14.4.6 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_prefetch\_register ( **amd\_dbgapi\_process\_id\_t** *process\_id*, **amd\_dbgapi\_wave\_id\_t** *wave\_id*, **amd\_dbgapi\_register\_id\_t** *register\_id*, **amd\_dbgapi\_size\_t** *register\_count* )

Prefetch register values.

A hint to indicate that a range of registers may be read using [amd\\_dbgapi\\_read\\_register](#) in the future. This can improve the performance of reading registers as the library may be able to batch the prefetch requests into one request.

The wave must be stopped. If the wave is resumed, then any prefetch requests for registers that were not subsequently read may be discarded and so provide no performance benefit. Prefetch requests for registers that are never subsequently read may in fact reduce performance.

The registers to prefetch are specified as the first register and the number of registers. The first register can be any register supported by the wave. The number of registers is in terms of the wave register order returned by [amd\\_dbgapi\\_wave\\_register\\_list](#). If the number exceeds the number of wave registers, then only up to the last wave register is prefetched.

The register handle must be valid for the architecture, and the wave must have allocated that register.

## Parameters

in	<i>process_id</i>	The process to which the wave belongs.
in	<i>wave_id</i>	The wave being queried for the register.
in	<i>register_id</i>	The first register being requested.
in	<i>register_count</i>	The number of registers being requested.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully. Registers may be prefetched.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. No registers are prefetched.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<i>wave_id</i> is invalid. No registers are prefetched.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID</a>	<i>register_id</i> is invalid for the architecture of <i>wave_id</i> , or not allocated for <i>wave_id</i> . <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED</a>	<i>wave_id</i> is not stopped. No registers are prefetched.

2.14.4.7 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_read\_register ( **amd\_dbgapi\_process\_id\_t** *process\_id*, **amd\_dbgapi\_wave\_id\_t** *wave\_id*, **amd\_dbgapi\_register\_id\_t** *register\_id*, **amd\_dbgapi\_size\_t** *offset*, **amd\_dbgapi\_size\_t** *value\_size*, void \* *value* )

Read a register.

*value\_size* bytes are read from the register starting at *offset* into *value*.

The wave must be stopped.

The register handle must be valid for the architecture, and the wave must have allocated that register.

The size of the register can vary depending on the wave. The register size can be obtained using [amd\\_dbgapi\\_wave\\_register\\_get\\_info](#) with the [AMD\\_DBGAPI\\_REGISTER\\_INFO\\_SIZE](#) query.

## Parameters

in	<i>process_id</i>	The process to which the wave belongs.
in	<i>wave_id</i>	The wave to being queried for the register.
in	<i>register_id</i>	The register being requested.
in	<i>offset</i>	The first byte to start reading the register. The offset is zero based starting from the least significant byte of the register.
in	<i>value_size</i>	The number of bytes to read from the register which must be greater than 0 and less than the size of the register minus <i>offset</i> .



out	value	The bytes read from the register. Must point to an array of at least value_size bytes.
-----	-------	--

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and value is set to value_size bytes starting at offset from the contents of the register.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	process_id is invalid. value are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	wave_id is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID</a>	register_id is invalid for the architecture of wave_id, or not allocated for wave_id. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED</a>	wave_id is not stopped. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	value is NULL. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	value_size is 0 or greater than the size of the register minus offset. value is unaltered.

2.14.4.8 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_register\_is\_in\_register\_class ( amd\_dbgapi\_architecture\_id\_t architecture\_id, amd\_dbgapi\_register\_id\_t register\_id, amd\_dbgapi\_register\_class\_id\_t register\_class\_id, amd\_dbgapi\_register\_class\_state\_t \* register\_class\_state )

Determine if a register is a member of a register class.

The register and register class must both belong to the same architecture.

## Parameters

in	architecture_id	The architecture of the register class and register.
in	register_id	The handle of the register being queried.
in	register_class_id	The handle of the register class being queried.
out	register_class_state	<a href="#">AMD_DBGAPI_REGISTER_CLASS_STATE_NOT_MEMBER</a> if the register is not in the register class. <a href="#">AMD_DBGAPI_REGISTER_CLASS_STATE_MEMBER</a> if the register is in the register class.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in register_class_state.
---	---

<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized and <code>register_class_state</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and <code>register_class_state</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</i></a>	<code>architecture_id</code> is invalid. <code>register_class_state</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID</i></a>	<code>register_id</code> is invalid for <code>architecture_id</code> . <code>register_class_state</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_CLASS_ID</i></a>	<code>register_class_id</code> is invalid for <code>architecture_id</code> . <code>register_class_state</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</i></a>	<code>register_class_state</code> is NULL. <code>register_class_state</code> is unaltered.

2.14.4.9 `amd_dbgapi_status_t` `AMD_DBGAPI amd_dbgapi_wave_register_get_info ( amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_register_id_t register_id, amd_dbgapi_register_info_t query, size_t value_size, void * value )`

Query information about a register of a wave.

`amd_dbgapi_register_info_t` specifies the queries supported and the type returned using the `value` argument.

#### Parameters

in	<i>process_id</i>	The process to which the wave belongs.
in	<i>wave_id</i>	The wave to which the register belongs.
in	<i>register_id</i>	The handle of the register being queried.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <code>value</code> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

#### Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	The function has been executed successfully and the result is stored in <code>value</code> .
<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</i></a>	<code>process_id</code> is invalid. <code>value</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</i></a>	<code>wave_id</code> is invalid. <code>value</code> is unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID</a>	register_id is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	value is NULL or query is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	value_size does not match the size of the result. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate value returns NULL. value is unaltered.

2.14.4.10 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_wave\_register\_list ( **amd\_dbgapi\_process\_id\_t** process\_id, **amd\_dbgapi\_wave\_id\_t** wave\_id, **size\_t**\* register\_count, **amd\_dbgapi\_register\_id\_t**\*\* registers )

Report the list of registers supported by a wave.

This list is the registers allocated for a specific wave and may not be all the registers supported by the architecture. For example, AMD GPU architectures can specify the number of vector and scalar registers when a wave is created. Use the [amd\\_dbgapi\\_architecture\\_register\\_list](#) operation to determine the full set of registers supported by the architecture.

The order of the register handles in the list is stable between calls. It is equal to, or a subset of, those returned by [amd\\_dbgapi\\_architecture\\_register\\_list](#) and in the same order.

#### Parameters

in	process_id	The process to which the wave belongs.
in	wave_id	The wave being queried.
out	register_count	The number of wave registers.
out	registers	A pointer to an array of <a href="#">amd_dbgapi_register_id_t</a> with register_count elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in register_count and registers.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and register_count and registers are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and register_count and registers are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	architecture_id is invalid. register_count and registers are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	register_count or registers are NULL. register_count and registers are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate registers returns NULL. register_count and registers are unaltered.

2.14.4.11 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_write_register ( amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_register_id_t register_id, amd_dbgapi_size_t offset, amd_dbgapi_size_t value_size, const void * value )`

Write a register.

`value_size` bytes are written into the register starting at `offset`.

The wave must be stopped.

The register handle must be valid for the architecture, and the wave must have allocated that register.

The size of the register can vary depending on the wave. The register size can be obtained using [amd\\_dbgapi\\_wave\\_register\\_get\\_info](#) with the [AMD\\_DBGAPI\\_REGISTER\\_INFO\\_SIZE](#) query.

#### Parameters

in	<i>process_id</i>	The process to which the wave belongs.
in	<i>wave_id</i>	The wave to being queried for the register.
in	<i>register_id</i>	The register being requested.
in	<i>offset</i>	The first byte to start writing the register. The offset is zero based starting from the least significant byte of the register.
in	<i>value_size</i>	The number of bytes to write to the register which must be greater than 0 and less than the size of the register minus <code>offset</code> .
in	<i>value</i>	The bytes to write to the register. Must point to an array of at least <code>value_size</code> bytes.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and <code>value_size</code> bytes have been written to the contents of the register starting at <code>offset</code> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized. The register is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized. The register is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. The register is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<code>wave_id</code> is invalid. The register is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID</a>	<code>register_id</code> is invalid for the architecture of <code>wave_id</code> , or not allocated for <code>wave_id</code> . <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED</a>	<code>wave_id</code> is not stopped. The register is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>value</code> is NULL. The register is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<code>value_size</code> is 0 or greater than the size of the register minus <code>offset</code> . The register is unaltered.

## 2.15 Memory

Operations related to AMD GPU memory access.

### Data Structures

- struct `amd_dbgapi_address_class_id_t`  
*Opaque source language address class handle.*
- struct `amd_dbgapi_address_space_id_t`  
*Opaque address space handle.*

### Macros

- #define `AMD_DBGAPI_LANE_NONE` (`amd_dbgapi_lane_id_t` (-1))  
*The NULL lane handle.*
- #define `AMD_DBGAPI_ADDRESS_CLASS_NONE` (`amd_dbgapi_address_class_id_t`{ 0 })  
*The NULL address class handle.*

### Typedefs

- typedef `uint32_t` `amd_dbgapi_lane_id_t`  
*A wave lane handle.*
- typedef `uint64_t` `amd_dbgapi_segment_address_t`  
*Each address space has its own linear address to access it termed a segment address.*

### Enumerations

- enum `amd_dbgapi_address_class_info_t` { `AMD_DBGAPI_ADDRESS_CLASS_INFO_NAME` = 1, `AMD_DBGAPI_ADDRESS_CLASS_INFO_ADDRESS_SPACE` = 2 }  
*Source language address class queries that are supported by `amd_dbgapi_architecture_address_class_get_info`.*
- enum `amd_dbgapi_address_space_access_t` { `AMD_DBGAPI_ADDRESS_SPACE_ACCESS_ALL` = 1, `AMD_DBGAPI_ADDRESS_SPACE_ACCESS_PROGRAM_CONSTANT` = 2, `AMD_DBGAPI_ADDRESS_SPACE_ACCESS_DISPATCH_CONSTANT` = 3 }  
*Indication of how the address space is accessed.*
- enum `amd_dbgapi_address_space_info_t` { `AMD_DBGAPI_ADDRESS_SPACE_INFO_NAME` = 1, `AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE` = 2, `AMD_DBGAPI_ADDRESS_SPACE_INFO_NULL_ADDRESS` = 3, `AMD_DBGAPI_ADDRESS_SPACE_INFO_ACCESS` = 4 }  
*Address space queries that are supported by `amd_dbgapi_address_space_get_info`.*
- enum `amd_dbgapi_address_space_alias_t` { `AMD_DBGAPI_ADDRESS_SPACE_ALIAS_NONE` = 0, `AMD_DBGAPI_ADDRESS_SPACE_ALIAS_MAY` = 1 }  
*Indication of whether addresses in two address spaces may alias.*
- enum `amd_dbgapi_address_class_state_t` { `AMD_DBGAPI_ADDRESS_CLASS_STATE_NOT_MEMBER` = 0, `AMD_DBGAPI_ADDRESS_CLASS_STATE_MEMBER` = 1 }  
*Indication of whether a segment address in an address space is a member of an source language address class.*
- enum `amd_dbgapi_memory_precision_t` { `AMD_DBGAPI_MEMORY_PRECISION_NONE` = 0, `AMD_DBGAPI_MEMORY_PRECISION_PRECISE` = 1 }  
*Memory access precision.*

## Functions

- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_architecture\\_address\\_class\\_get\\_info](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_address\\_class\\_id\\_t](#) address\_class\_id, [amd\\_dbgapi\\_address\\_class\\_info\\_t](#) query, [size\\_t](#) value\_size, void \*value)  
*Query information about a source language address class of an architecture.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_architecture\\_address\\_class\\_list](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [size\\_t](#) \*address\_class\_count, [amd\\_dbgapi\\_address\\_class\\_id\\_t](#) \*\*address\_classes)  
*Report the list of source language address classes supported by the architecture.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_dwarf\\_address\\_class\\_to\\_address\\_class](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [uint64\\_t](#) dwarf\_address\_class, [amd\\_dbgapi\\_address\\_class\\_id\\_t](#) \*address\_class\_id)  
*Return the architecture source language address class from a DWARF address class number.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_address\\_space\\_get\\_info](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id, [amd\\_dbgapi\\_address\\_space\\_info\\_t](#) query, [size\\_t](#) value\_size, void \*value)  
*Query information about an address space.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_architecture\\_address\\_space\\_list](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [size\\_t](#) \*address\_space\_count, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) \*\*address\_spaces)  
*Report the list of address spaces supported by the architecture.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_dwarf\\_address\\_space\\_to\\_address\\_space](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [uint64\\_t](#) dwarf\_address\_space, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) \*address\_space\_id)  
*Return the address space from an AMD GPU DWARF address space number.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_address\\_spaces\\_may\\_alias](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id1, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id2, [amd\\_dbgapi\\_address\\_space\\_alias\\_t](#) \*address\_space\_alias)  
*Determine if an address in one address space may alias an address in another address space.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_convert\\_address\\_space](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_lane\\_id\\_t](#) lane\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) source\_address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) source\_segment\_address, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) destination\_address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) \*destination\_segment\_address)  
*Convert a source segment address in the source address space into a destination segment address in the destination address space.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_address\\_is\\_in\\_address\\_class](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_lane\\_id\\_t](#) lane\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) segment\_address, [amd\\_dbgapi\\_address\\_class\\_id\\_t](#) address\_class\_id, [amd\\_dbgapi\\_address\\_class\\_state\\_t](#) \*address\_class\_state)  
*Determine if a segment address in an address space is a member of a source language address class.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_read\\_memory](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_lane\\_id\\_t](#) lane\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) segment\_address, [amd\\_dbgapi\\_size\\_t](#) \*value\_size, void \*value)  
*Read memory.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_write\\_memory](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_lane\\_id\\_t](#) lane\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) segment\_address, [amd\\_dbgapi\\_size\\_t](#) \*value\_size, const void \*value)  
*Write memory.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_set\\_memory\\_precision](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_agent\\_id\\_t](#) agent\_id, [amd\\_dbgapi\\_memory\\_precision\\_t](#) memory\_precision)  
*Control precision of memory access reporting.*

### 2.15.1 Detailed Description

Operations related to AMD GPU memory access. The AMD GPU supports allocating memory in different address spaces. See [User Guide for AMDGPU Backend - LLVM - Address Spaces](#).

### 2.15.2 Macro Definition Documentation

2.15.2.1 `#define AMD_DBGAPI_ADDRESS_CLASS_NONE (amd_dbgapi_address_class_id_t{ 0 })`

The NULL address class handle.

2.15.2.2 `#define AMD_DBGAPI_LANE_NONE (amd_dbgapi_lane_id_t (-1))`

The NULL lane handle.

### 2.15.3 Typedef Documentation

2.15.3.1 `typedef uint32_t amd_dbgapi_lane_id_t`

A wave lane handle.

A wave can have one or more lanes controlled by an execution mask. Vector instructions will be performed for each lane of the wave that the execution mask has enabled. Vector instructions can access registers that are vector registers. A vector register has a separate value for each lane, and vector instructions will access the corresponding component for each lane's evaluation of the instruction.

The number of lanes of a wave can be obtained with the [AMD\\_DBGAPI\\_WAVE\\_INFO\\_LANE\\_COUNT](#) query. Different waves of the same architecture can have different lane counts.

The AMD GPU compiler may map source language threads of execution to lanes of a wave. The DWARF debug information which maps such source languages to the generate architecture specific code must include information about the lane mapping.

The `::AMD_DBGAPI_ADDRESS_SPACE_LANE` address space supports memory allocated independently for each lane of a wave.

Lanes are numbered from 0 to [AMD\\_DBGAPI\\_WAVE\\_INFO\\_LANE\\_COUNT](#) minus 1.

Only unique for a single wave of a single process.

2.15.3.2 `typedef uint64_t amd_dbgapi_segment_address_t`

Each address space has its own linear address to access it termed a segment address.

Different address spaces may have memory locations that alias each other, but the segment address for such memory locations may be different in each address space. Consequently a segment address is specific to an address space.

Some address spaces may access memory that is allocated independently for each work-group, for each wave, or for each lane of a wave. Consequently a segment address may be specific to a wave or lane of a wave.

See [User Guide for AMDGPU Backend - LLVM - Address Spaces](#).

### 2.15.4 Enumeration Type Documentation

#### 2.15.4.1 enum amd\_dbgapi\_address\_class\_info\_t

Source language address class queries that are supported by [amd\\_dbgapi\\_architecture\\_address\\_class\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_architecture\\_address\\_class\\_get\\_info](#).

##### Enumerator

**AMD\_DBGAPI\_ADDRESS\_CLASS\_INFO\_NAME** Return the source language address class name. The type of this attribute is a pointer to a NUL terminated `char`. It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

**AMD\_DBGAPI\_ADDRESS\_CLASS\_INFO\_ADDRESS\_SPACE** Return the architecture specific address space that is used to implement a pointer or reference to the source language address class. The type of this attribute is [amd\\_dbgapi\\_address\\_class\\_id\\_t](#).

See [User Guide for AMDGPU Backend - Code Object - DWARF - Address Class Mapping](#).

#### 2.15.4.2 enum amd\_dbgapi\_address\_class\_state\_t

Indication of whether a segment address in an address space is a member of an source language address class.

##### Enumerator

**AMD\_DBGAPI\_ADDRESS\_CLASS\_STATE\_NOT\_MEMBER** The segment address in the address space is not a member of the source language address class.

**AMD\_DBGAPI\_ADDRESS\_CLASS\_STATE\_MEMBER** The segment address in the address space is a member of the source language address class.

#### 2.15.4.3 enum amd\_dbgapi\_address\_space\_access\_t

Indication of how the address space is accessed.

##### Enumerator

**AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_ALL** The address space supports all accesses. Values accessed can change during the lifetime of the program.

**AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_PROGRAM\_CONSTANT** The address space is read only. Values accessed are always the same value for the lifetime of the program execution.

**AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_DISPATCH\_CONSTANT** The address space is only read the waves of a kernel dispatch. Values accessed are always the same value for the lifetime of the dispatch.

#### 2.15.4.4 enum amd\_dbgapi\_address\_space\_alias\_t

Indication of whether addresses in two address spaces may alias.

##### Enumerator

**AMD\_DBGAPI\_ADDRESS\_SPACE\_ALIAS\_NONE** No addresses in the address spaces can alias.

**AMD\_DBGAPI\_ADDRESS\_SPACE\_ALIAS\_MAY** Addresses in the address spaces may alias.



2.15.4.5 `enum amd_dbgapi_address_space_info_t`

Address space queries that are supported by [amd\\_dbgapi\\_address\\_space\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_address\\_space\\_get\\_info](#).

## Enumerator

**AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_NAME** Return the address space name. The type of this attribute is a pointer to a NUL terminated `char*`. It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

**AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_ADDRESS\_SIZE** Return the byte size of an address in the address space. The type of this attribute is [amd\\_dbgapi\\_size\\_t](#).

**AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_NULL\_ADDRESS** Return the NULL segment address value in the address space. The type of this attribute is `amd_dbgapi_segment_address_t`.

**AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_ACCESS** Return the address space access. The type of this attribute is `uint32_t` with values defined by [amd\\_dbgapi\\_address\\_space\\_access\\_t](#).

2.15.4.6 `enum amd_dbgapi_memory_precision_t`

Memory access precision.

The AMD GPU can overlap the execution of memory instructions with other instructions. This can result in a wave stopping due to a memory violation or hardware data watchpoint hit with a program counter beyond the instruction that caused the wave to stop.

Some architectures allow the hardware to be configured to always wait for memory operations to complete before continuing. This will result in the wave stopping at the instruction immediately after the one that caused the stop event. Enabling this mode can make execution of waves significantly slower.

The [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_PRECISE\\_MEMORY\\_SUPPORTED](#) query can be used to determine if an architecture supports controlling precise memory accesses.

## Enumerator

**AMD\_DBGAPI\_MEMORY\_PRECISION\_NONE** Memory instructions execute normally and a wave does not wait for the memory access to complete.

**AMD\_DBGAPI\_MEMORY\_PRECISION\_PRECISE** A wave waits for memory instructions to complete before executing further instructions. This can cause a wave to execute significantly slower.

## 2.15.5 Function Documentation

2.15.5.1 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_address_is_in_address_class( amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_lane_id_t lane_id, amd_dbgapi_address_space_id_t address_space_id, amd_dbgapi_segment_address_t segment_address, amd_dbgapi_address_class_id_t address_class_id, amd_dbgapi_address_class_state_t * address_class_state )`

Determine if a segment address in an address space is a member of a source language address class.

The address space and source language address class must both belong to the same architecture.

The address space, source language address class, and wave must all belong to the same architecture.

## Parameters

in	<i>process_id</i>	The process to which the <i>wave_id</i> belongs.
in	<i>wave_id</i>	The wave that is using the address.
in	<i>lane_id</i>	The lane of the <i>wave_id</i> that is using the address.
in	<i>address_space_id</i>	The address space of the <i>segment_address</i> . If the address space is dependent on: the active lane then the <i>lane_id</i> with in the <i>wave_id</i> is used; the active work-group then the work-group of <i>wave_id</i> is used; or the active wave then the <i>wave_id</i> is used.
in	<i>segment_address</i>	The integral value of the segment address. Only the bits corresponding to the address size for the <i>address_space</i> requested are used. The address size is provided by the <a href="#">AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE</a> query.
in	<i>address_class_id</i>	The handle of the source language address class.
out	<i>address_class_state</i>	<a href="#">AMD_DBGAPI_ADDRESS_CLASS_STATE_NOT_MEMBER</a> if the address is not in the address class. <a href="#">AMD_DBGAPI_ADDRESS_CLASS_STATE_MEMBER</a> if the address is in the address class.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>address_class_state</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>address_class_state</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>address_class_state</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. <i>address_class_state</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<i>wave_id</i> is invalid. <i>address_class_state</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_LANE_ID</a>	<i>lane_id</i> is invalid. <i>address_class_state</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID</a>	<i>address_space_id</i> is invalid for the architecture of <i>wave_id</i> . <i>address_class_state</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_CLASS_ID</a>	<i>address_class_id</i> is invalid for the architecture of <i>wave_id</i> . <i>address_class_state</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>address_class_state</i> is NULL. <i>address_class_state</i> is unaltered.

2.15.5.2 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_address_space_get_info ( amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_address_space_id_t address_space_id, amd_dbgapi_address_space_info_t query, size_t value_size, void * value )`

Query information about an address space.

[amd\\_dbgapi\\_address\\_space\\_info\\_t](#) specifies the queries supported and the type returned using the *value* argument.

## Parameters

in	<i>architecture_id</i>	The architecture of the address space.
in	<i>address_space_id</i>	The address space.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <i>value</i> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>value</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<i>architecture_id</i> is invalid. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID</a>	<i>address_space_id</i> is invalid for <i>architecture_id</i> . <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>query</i> is invalid or <i>value</i> is NULL. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<i>value_size</i> does not match the size of the result. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <i>value</i> returns NULL. <i>value</i> is unaltered.

**2.15.5.3** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_address_spaces_may_alias ( amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_address_space_id_t address_space_id1, amd_dbgapi_address_space_id_t address_space_id2, amd_dbgapi_address_space_alias_t * address_space_alias )`

Determine if an address in one address space may alias an address in another address space.

The address spaces must match the architecture.

## Parameters

in	<i>architecture_id</i>	The architecture to which the address spaces belong.
in	<i>address_space_id1</i>	An address space.
in	<i>address_space_id2</i>	An address space.
out	<i>address_space_alias</i>	<a href="#">AMD_DBGAPI_ADDRESS_SPACE_ALIAS_NONE</a> if the address spaces do not alias. <a href="#">AMD_DBGAPI_ADDRESS_SPACE_ALIAS_MAY</a> if the address spaces may alias.

## Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	The function has been executed successfully and the result is stored in <code>address_space_alias</code> .
<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized and <code>address_space_alias</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and <code>address_space_alias</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</i></a>	<code>architecture_id</code> is invalid. <code>address_space_alias</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID</i></a>	<code>address_space_id1</code> or <code>address_space_id2</code> are invalid for <code>architecture_id</code> . <code>address_space_alias</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</i></a>	<code>address_space_alias</code> is NULL. <code>address_space_alias</code> is unaltered.

2.15.5.4 `amd_dbgapi_status_t` `AMD_DBGAPI` `amd_dbgapi_architecture_address_class_get_info` (  
`amd_dbgapi_architecture_id_t` `architecture_id`, `amd_dbgapi_address_class_id_t` `address_class_id`,  
`amd_dbgapi_address_class_info_t` `query`, `size_t` `value_size`, `void *` `value` )

Query information about a source language address class of an architecture.

`amd_dbgapi_address_class_info_t` specifies the queries supported and the type returned using the `value` argument.

## Parameters

in	<code>architecture_id</code>	The architecture to which the source language address class belongs.
in	<code>address_class_id</code>	The handle of the source language address class being queried.
in	<code>query</code>	The query being requested.
in	<code>value_size</code>	Size of the memory pointed to by <code>value</code> . Must be equal to the byte size of the query result.
out	<code>value</code>	Pointer to memory where the query result is stored.

## Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	The function has been executed successfully and the result is stored in <code>value</code> .
<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and <code>value</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</i></a>	<code>architecture_id</code> is invalid. <code>value</code> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_CLASS_ID</i></a>	<code>address_class_id</code> is invalid for the architecture of <code>architecture_id</code> . <code>value</code> is unaltered.

<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	value is NULL or query is invalid. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	value_size does not match the size of the result. value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate value returns NULL. value is unaltered.

2.15.5.5 **amd\_dbgapi\_status\_t** **AMD\_DBGAPI** amd\_dbgapi\_architecture\_address\_class\_list ( **amd\_dbgapi\_architecture\_id\_t** architecture\_id, **size\_t** \* address\_class\_count, **amd\_dbgapi\_address\_class\_id\_t** \*\* address\_classes )

Report the list of source language address classes supported by the architecture.

The order of the source language address class handles in the list is stable between calls.

#### Parameters

in	<i>architecture_id</i>	The architecture being queried.
out	<i>address_class_count</i>	The number of architecture source language address classes.
out	<i>address_classes</i>	A pointer to an array of <a href="#">amd_dbgapi_address_class_id_t</a> with <i>address_class_count</i> elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>address_class_count</i> and <i>address_classes</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>address_class_count</i> and <i>address_classes</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>address_class_count</i> and <i>address_classes</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<i>architecture_id</i> is invalid. <i>address_class_count</i> and <i>address_classes</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>address_class_count</i> or <i>address_classes</i> are NULL. <i>address_class_count</i> and <i>address_classes</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <i>address_classes</i> returns NULL. <i>address_class_count</i> and <i>address_classes</i> are unaltered.

2.15.5.6 **amd\_dbgapi\_status\_t** **AMD\_DBGAPI** amd\_dbgapi\_architecture\_address\_space\_list ( **amd\_dbgapi\_architecture\_id\_t** architecture\_id, **size\_t** \* address\_space\_count, **amd\_dbgapi\_address\_space\_id\_t** \*\* address\_spaces )

Report the list of address spaces supported by the architecture.

The order of the address space handles in the list is stable between calls.

## Parameters

in	<i>architecture_id</i>	The architecture being queried.
out	<i>address_space_count</i>	The number of architecture address spaces.
out	<i>address_spaces</i>	A pointer to an array of <a href="#">amd_dbgapi_address_space_id_t</a> with <i>address_space_count</i> elements. It is allocated by the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback and is owned by the client.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>address_space_count</i> and <i>address_spaces</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>address_space_count</i> and <i>address_spaces</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>address_space_count</i> and <i>address_spaces</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</a>	<i>architecture_id</i> is invalid. <i>address_space_count</i> and <i>address_spaces</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>address_space_count</i> and <i>address_spaces</i> are NULL. <i>address_space_count</i> and <i>address_spaces</i> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <i>address_spaces</i> returns NULL. <i>address_space_count</i> and <i>address_spaces</i> are unaltered.

**2.15.5.7** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_convert_address_space ( amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_lane_id_t lane_id, amd_dbgapi_address_space_id_t source_address_space_id, amd_dbgapi_segment_address_t source_segment_address, amd_dbgapi_address_space_id_t destination_address_space_id, amd_dbgapi_segment_address_t * destination_segment_address )`

Convert a source segment address in the source address space into a destination segment address in the destination address space.

If the source segment address is the NULL value in the source address space then it is converted to the NULL value in the destination address space. The NULL address is provided by the [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_INFO\\_NULL\\_ADDRESS](#) query.

An error is returned if the source segment address has no corresponding segment address in the destination address space. The source and destination address spaces must have the same linear ordering. For example, a swizzled address space is not the same linear ordering as an unswizzled address space. The source and destination address spaces must either both depend on the active lane, both depend on the same lane, or both not depend on the lane.

## Parameters

in	<i>process_id</i>	The process to which the <i>wave_id</i> belongs.
in	<i>wave_id</i>	The wave that is using the address.
in	<i>lane_id</i>	The lane of the <i>wave_id</i> that is using the address.
in	<i>source_address_space</i>	The address space of the <i>source_segment_address</i> .

in	<i>source_segment_address</i>	The integral value of the source segment address. Only the bits corresponding to the address size for the <i>source_address_space</i> requested are used. The address size is provided by the <a href="#">AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE</a> query.
in	<i>destination_address_space</i>	The address space to which to convert <i>source_segment_address</i> that is in <i>source_address_space</i> .
out	<i>destination_segment_address</i>	The integral value of the segment address in <i>destination_address_space</i> that corresponds to <i>source_segment_address</i> in <i>source_address_space</i> . The bits corresponding to the address size for the <i>destination_address_space</i> are updated, and any remaining bits are set to zero. The address size is provided by the <a href="#">AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE</a> query.

## Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>destination_segment_address</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>destination_segment_address</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>destination_segment_address</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<i>process_id</i> is invalid. <i>destination_segment_address</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<i>wave_id</i> is invalid. <i>destination_segment_address</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_LANE_ID</a>	<i>lane_id</i> is invalid, or <i>lane_id</i> is <a href="#">AMD_DBGAPI_LANE_NONE</a> and <i>source_address_space</i> depends on the active lane. <i>destination_segment_address</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID</a>	<i>source_address_space_id</i> or <i>destination_address_space_id</i> are invalid for the architecture of <i>wave_id</i> . value is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_CONVERSION</a>	The <i>source_segment_address</i> in the <i>source_address_space_id</i> is not an address that can be represented in the <i>destination_address_space_id</i> . <i>destination_segment_address</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>destination_segment_address</i> is NULL. <i>destination_segment_address</i> is unaltered.

2.15.5.8 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_dwarf_address_class_to_address_class ( amd_dbgapi_architecture_id_t architecture_id, uint64_t dwarf_address_class, amd_dbgapi_address_class_id_t * address_class_id )`

Return the architecture source language address class from a DWARF address class number.

See [User Guide for AMDGPU Backend - Code Object - DWARF - Address Class Mapping](#).

## Parameters

in	<i>architecture_id</i>	The architecture of the source language address class.
----	------------------------	--

in	<i>dwarf_address_class</i>	The DWARF source language address class.
out	<i>address_class_id</i>	The source language address class that corresponds to the DWARF address class for the architecture.

## Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	The function has been executed successfully and the result is stored in <i>address_class_id</i> .
<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized and <i>address_class_id</i> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and <i>address_class_id</i> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</i></a>	<i>architecture_id</i> is invalid. <i>address_class_id</i> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</i></a>	<i>dwarf_address_class</i> is not valid for <i>architecture_id</i> or <i>address_class_id</i> is NULL. <i>address_class_id</i> is unaltered.

2.15.5.9 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_dwarf\_address\_space\_to\_address\_space ( **amd\_dbgapi\_architecture\_id\_t** *architecture\_id*, **uint64\_t** *dwarf\_address\_space*, **amd\_dbgapi\_address\_space\_id\_t** \* *address\_space\_id* )

Return the address space from an AMD GPU DWARF address space number.

A DWARF address space describes the architecture specific address spaces. If is used in DWARF location expressions that calculate addresses. See [User Guide for AMDGPU Backend - Code Object - DWARF - Address Space Mapping](#).

## Parameters

in	<i>architecture_id</i>	The architecture of the address space.
in	<i>dwarf_address_space</i>	The AMD GPU DWARF address space.
out	<i>address_space_id</i>	The address space that corresponds to the DWARF address space for the architecture <i>architecture_id</i> .

## Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	The function has been executed successfully and the result is stored in <i>address_space_id</i> .
<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized and <i>address_space_id</i> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized and <i>address_space_id</i> is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID</i></a>	<i>architecture_id</i> is invalid. <i>address_space_id</i> is unaltered.



<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	dwarf_address_space is not valid for architecture_id, or address_space_id is NULL. address_space_id is unaltered.
--	---

2.15.5.10 **amd\_dbgapi\_status\_t** AMD\_DBGAPI amd\_dbgapi\_read\_memory ( **amd\_dbgapi\_process\_id\_t** *process\_id*, **amd\_dbgapi\_wave\_id\_t** *wave\_id*, **amd\_dbgapi\_lane\_id\_t** *lane\_id*, **amd\_dbgapi\_address\_space\_id\_t** *address\_space\_id*, **amd\_dbgapi\_segment\_address\_t** *segment\_address*, **amd\_dbgapi\_size\_t** \* *value\_size*, void \* *value* )

Read memory.

The memory bytes in *address\_space* are read starting at *segment\_address* sequentially into *value* until *value\_size* bytes have been read or an invalid memory address is reached. *value\_size* is set to the number of bytes read successfully.

The wave must be stopped.

The library performs all necessary hardware cache management so that the memory values read are coherent with the *wave\_id*.

#### Parameters

in	<i>process_id</i>	The process to which the <i>wave_id</i> belongs.
in	<i>wave_id</i>	The wave that is reading the memory.
in	<i>lane_id</i>	The lane of <i>wave_id</i> that is accessing the memory. If the <i>address_space</i> does not depend on the active lane then this is ignored and may be <a href="#">AMD_DBGAPI_LANE_NONE</a> .
in	<i>address_space_id</i>	The address space of the <i>segment_address</i> . If the address space is dependent on: the active lane then the <i>lane_id</i> with in the <i>wave_id</i> is used; the active work-group then the work-group of <i>wave_id</i> is used; or the active wave then the <i>wave_id</i> is used.
in	<i>segment_address</i>	The integral value of the segment address. Only the bits corresponding to the address size for the <i>address_space</i> requested are used. The address size is provided by the <a href="#">AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE</a> query.
in,out	<i>value_size</i>	Pass in the number of bytes to read from memory. Return the number of bytes successfully read from memory.
out	<i>value</i>	Pointer to memory where the result is stored. Must be an array of at least input <i>value_size</i> bytes.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	Either the input <i>value_size</i> was 0, or the input <i>value_size</i> was greater than 0 and one or more bytes have been read successfully. The output <i>value_size</i> is set to the number of bytes successfully read, which will be 0 if the input <i>value_size</i> was 0. The first output <i>value_size</i> bytes of <i>value</i> are set to the bytes successfully read, all other bytes in <i>value</i> are unaltered.
---	--

<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized. <code>value_size</code> and <code>value</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized. <code>value_size</code> and <code>value</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	<code>process_id</code> is invalid. <code>value_size</code> and <code>value</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</a>	<code>wave_id</code> is invalid. <code>value_size</code> and <code>value</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_LANE_ID</a>	<code>lane_id</code> is invalid, or <code>lane_id</code> is <a href="#">AMD_DBGAPI_LANE_NONE</a> and <code>address_space</code> depends on the active lane. <code>value_size</code> and <code>value</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID</a>	<code>address_space_id</code> is invalid for the architecture of <code>wave_id</code> . <code>value</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED</a>	<code>wave_id</code> is not stopped. <code>value_size</code> and <code>value</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>value</code> or <code>value_size</code> are NULL. <code>value_size</code> and <code>value</code> are unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_MEMORY_ACCESS</a>	The input <code>value_size</code> was greater than 0 and no bytes were successfully read. The output <code>value_size</code> is set to 0. All bytes in <code>value</code> are unaltered.

2.15.5.11 **amd\_dbgapi\_status\_t** **AMD\_DBGAPI** **amd\_dbgapi\_set\_memory\_precision** ( **amd\_dbgapi\_process\_id\_t** *process\_id*, **amd\_dbgapi\_agent\_id\_t** *agent\_id*, **amd\_dbgapi\_memory\_precision\_t** *memory\_precision* )

Control precision of memory access reporting.

An agent can be set to [AMD\\_DBGAPI\\_MEMORY\\_PRECISION\\_NONE](#) to disable precise memory reporting. Use the [AMD\\_DBGAPI\\_ARCHITECTURE\\_INFO\\_PRECISE\\_MEMORY\\_SUPPORTED](#) query to determine if an agent's architecture supports another memory precision.

The memory precision is set independently for each agent, and only affects the waves executing on that agent.

#### Parameters

in	<i>process_id</i>	The process being configured.
in	<i>agent_id</i>	The agent to configure.
in	<i>memory_precision</i>	The memory precision to set.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the agent has been configured.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and no agent configuration is changed.

<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	process_id is invalid. No agent configuration is changed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_AGENT_ID</a>	agent_id is invalid. No agent configuration is changed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_SUPPORTED</a>	The requested memory_precision is not supported for the architecture of the agent. No agent configuration is changed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	memory_precision is an invalid value. No agent configuration is changed.

2.15.5.12 `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_write_memory ( amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_lane_id_t lane_id, amd_dbgapi_address_space_id_t address_space_id, amd_dbgapi_segment_address_t segment_address, amd_dbgapi_size_t * value_size, const void * value )`

Write memory.

The memory bytes in address\_space are written starting at segment\_address sequentially from value until value\_size bytes have been written or an invalid memory address is reached. value\_size is set to the number of bytes written successfully.

The wave must be stopped.

The library performs all necessary hardware cache management so that the memory values written are coherent with the wave\_id.

#### Parameters

in	process_id	The process to which the wave_id belongs.
in	wave_id	The wave that is writing the memory.
in	lane_id	The lane of wave_id that is accessing the memory. If the address_space does not depend on the active lane then this is ignored and may be <a href="#">AMD_DBGAPI_LANE_NONE</a> .
in	address_space_id	The address space of the segment_address. If the address space is dependent on: the active lane then the lane_id with in the wave_id is used; the active work-group then the work-group of wave_id is used; or the active wave then the wave_id is used.
in	segment_address	The integral value of the segment address. Only the bits corresponding to the address size for the address_space requested are used. The address size is provided by the <a href="#">AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE</a> query.
in,out	value_size	Pass in the number of bytes to write to memory. Return the number of bytes successfully written to memory.
in	value	The bytes to write to memory. Must point to an array of at least input value_size bytes.

## Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	Either the input <code>value_size</code> was 0, or the input <code>value_size</code> was greater than 0 and one or more bytes have been written successfully. The output <code>value_size</code> is set to the number of bytes successfully written, which will be 0 if the input <code>value_size</code> was 0. The first output <code>value_size</code> bytes of memory starting at <code>segment_address</code> are updated, all other memory is unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_FATAL</i></a>	A fatal error occurred. The library is left uninitialized. The memory and <code>value_size</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i></a>	The library is not initialized. The library is left uninitialized. The memory and <code>value_size</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</i></a>	<code>process_id</code> is invalid. The memory and <code>value_size</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID</i></a>	<code>wave_id</code> is invalid. The memory and <code>value_size</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_LANE_ID</i></a>	<code>lane_id</code> is invalid, or <code>lane_id</code> is <a href="#"><i>AMD_DBGAPI_LANE_NONE</i></a> and <code>address_space</code> depends on the active lane. The memory and <code>value_size</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID</i></a>	<code>address_space_id</code> is invalid for the architecture of <code>wave_id</code> . The memory and <code>value_size</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED</i></a>	<code>wave_id</code> is not stopped. The memory and <code>value_size</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</i></a>	<code>value</code> or <code>value_size</code> are NULL. The memory and <code>value_size</code> are unaltered.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_MEMORY_ACCESS</i></a>	The input <code>value_size</code> was greater than 0 and no bytes were successfully written. The output <code>value_size</code> is set to 0. The memory is unaltered.

## 2.16 Events

Asynchronous event management.

### Data Structures

- struct `amd_dbgapi_event_id_t`  
*Opaque event handle.*

### Macros

- #define `AMD_DBGAPI_EVENT_NONE` (`amd_dbgapi_event_id_t{ 0 }`)  
*The NULL event handle.*

### Enumerations

- enum `amd_dbgapi_event_kind_t` {  
`AMD_DBGAPI_EVENT_KIND_NONE = 0`, `AMD_DBGAPI_EVENT_KIND_WAVE_STOP = 1`, `AMD_DBGAPI_EVENT_KIND_WAVE_COMMAND_TERMINATED = 2`, `AMD_DBGAPI_EVENT_KIND_CODE_OBJECT_LIST_UPDATED = 3`,  
`AMD_DBGAPI_EVENT_KIND_BREAKPOINT_RESUME = 4`, `AMD_DBGAPI_EVENT_KIND_RUNTIME = 5`, `AMD_DBGAPI_EVENT_KIND_QUEUE_ERROR = 6` }  
*The event kinds.*
- enum `amd_dbgapi_runtime_state_t` { `AMD_DBGAPI_RUNTIME_STATE_LOADED_SUPPORTED = 1`, `AMD_DBGAPI_RUNTIME_STATE_LOADED_UNsupported = 2`, `AMD_DBGAPI_RUNTIME_STATE_UNLOADED = 3` }  
*Inferior runtime state.*
- enum `amd_dbgapi_event_info_t` {  
`AMD_DBGAPI_EVENT_INFO_KIND = 1`, `AMD_DBGAPI_EVENT_INFO_WAVE = 2`, `AMD_DBGAPI_EVENT_INFO_BREAKPOINT = 3`, `AMD_DBGAPI_EVENT_INFO_CLIENT_THREAD = 4`,  
`AMD_DBGAPI_EVENT_INFO_RUNTIME_STATE = 5`, `AMD_DBGAPI_EVENT_INFO_RUNTIME_VERSION = 6`  
} }  
*Event queries that are supported by `amd_dbgapi_event_get_info`.*

### Functions

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_next_pending_event` (`amd_dbgapi_process_id_t` process\_id, `amd_dbgapi_event_id_t` \*event\_id, `amd_dbgapi_event_kind_t` \*kind)  
*Obtain the next pending event for a process.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_event_get_info` (`amd_dbgapi_process_id_t` process\_id, `amd_dbgapi_event_id_t` event\_id, `amd_dbgapi_event_info_t` query, `size_t` value\_size, void \*value)  
*Query information about an event.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_event_processed` (`amd_dbgapi_process_id_t` process\_id, `amd_dbgapi_event_id_t` event\_id)  
*Report that an event has been processed.*

### 2.16.1 Detailed Description

Asynchronous event management. Events can occur asynchronously. The library maintains a list of pending events that have happened but not yet been reported to the client. Events are maintained independently for each process.

When [amd\\_dbgapi\\_process\\_attach](#) successfully attaches to a process a [amd\\_dbgapi\\_notifier\\_t](#) notifier is created that is available using the [AMD\\_DBGAPI\\_PROCESS\\_INFO\\_NOTIFIER](#) query. When this indicates there may be pending events for the process, [amd\\_dbgapi\\_next\\_pending\\_event](#) can be used to retrieve the pending events.

The notifier must be reset before retrieving pending events so that the notifier will always conservatively indicate there may be pending events. After the client has processed an event it must report completion using [amd\\_dbgapi\\_event\\_processed](#).

See Also

[amd\\_dbgapi\\_notifier\\_t](#)

### 2.16.2 Macro Definition Documentation

#### 2.16.2.1 `#define AMD_DBGAPI_EVENT_NONE (amd_dbgapi_event_id_t{ 0 })`

The NULL event handle.

### 2.16.3 Enumeration Type Documentation

#### 2.16.3.1 `enum amd_dbgapi_event_info_t`

Event queries that are supported by [amd\\_dbgapi\\_event\\_get\\_info](#).

Each query specifies the type of data returned in the `value` argument to [amd\\_dbgapi\\_event\\_get\\_info](#).

Enumerator

**AMD\_DBGAPI\_EVENT\_INFO\_KIND** Return the event kind. The type of this attribute is [amd\\_dbgapi\\_event\\_kind\\_t](#).

**AMD\_DBGAPI\_EVENT\_INFO\_WAVE** Return the wave of a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) or [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_COMMAND\\_TERMINATED](#) event. The type of this attribute is a [amd\\_dbgapi\\_wave\\_id\\_t](#).

**AMD\_DBGAPI\_EVENT\_INFO\_BREAKPOINT** Return the breakpoint of a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_BREAKPOINT\\_RESUME](#) event. The type of this attribute is a [amd\\_dbgapi\\_breakpoint\\_id\\_t](#).

**AMD\_DBGAPI\_EVENT\_INFO\_CLIENT\_THREAD** Return the client thread of a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_BREAKPOINT\\_RESUME](#) event. The type of this attribute is a [amd\\_dbgapi\\_client\\_thread\\_id\\_t](#).

**AMD\_DBGAPI\_EVENT\_INFO\_RUNTIME\_STATE** Return if the runtime loaded in the inferior is supported by the library for a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_RUNTIME](#) event. The type of this attribute is `uint32_t` with a value defined by [amd\\_dbgapi\\_runtime\\_state\\_t](#).

**AMD\_DBGAPI\_EVENT\_INFO\_RUNTIME\_VERSION** Return the version of the runtime loaded in the inferior for a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_RUNTIME](#) event. The type of this attribute is a pointer to a NUL terminated `char*`. It is allocated by the [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) callback and is owned by the client.

#### 2.16.3.2 `enum amd_dbgapi_event_kind_t`

The event kinds.

## Enumerator

**AMD\_DBGAPI\_EVENT\_KIND\_NONE** No event.

**AMD\_DBGAPI\_EVENT\_KIND\_WAVE\_STOP** A wave has stopped.

**AMD\_DBGAPI\_EVENT\_KIND\_WAVE\_COMMAND\_TERMINATED** A command for a wave was not able to complete because the wave has terminated. Commands that can result in this event are [amd\\_dbgapi\\_wave\\_stop](#) and [amd\\_dbgapi\\_wave\\_resume](#) in single step mode. Since the wave terminated before stopping, this event will be reported instead of [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#).

The wave that terminated is available by the [AMD\\_DBGAPI\\_EVENT\\_INFO\\_WAVE](#) query. However, the wave will be invalid since it has already terminated. It is the client's responsibility to know what command was being performed and was unable to complete due to the wave terminating.

**AMD\_DBGAPI\_EVENT\_KIND\_CODE\_OBJECT\_LIST\_UPDATED** The list of code objects has changed. The thread that caused the code object list to change will be stopped until the event is reported as processed. Before reporting the event has been processed, the client must set any pending breakpoints for newly loaded code objects so that breakpoints will be set before any code in the code object is executed.

When the event is reported as complete, a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_BREAKPOINT\\_RESUME](#) event may be created which must be processed to resume the thread that caused the code object list to change. Leaving the thread stopped may prevent the inferior runtime from servicing requests from other threads.

**AMD\_DBGAPI\_EVENT\_KIND\_BREAKPOINT\_RESUME** Request to resume a host breakpoint. If [amd\\_dbgapi\\_report\\_breakpoint\\_hit](#) returns with `resume` as false then it indicates that events must be processed before the thread hitting the breakpoint can be resumed. When the necessary event(s) are reported as processed, this event will be added to the pending events. The breakpoint and client thread can then be queried by [amd\\_dbgapi\\_event\\_get\\_info](#) using [AMD\\_DBGAPI\\_EVENT\\_INFO\\_BREAKPOINT](#) and [AMD\\_DBGAPI\\_EVENT\\_INFO\\_CLIENT\\_THREAD](#) respectively. The client must then resume execution of the thread.

**AMD\_DBGAPI\_EVENT\_KIND\_RUNTIME** The runtime support in the inferior has been loaded or unloaded. Until it has been successfully loaded no code objects will be loaded and no waves will be created. The client can use this event to determine when to activate and deactivate AMD GPU debugging functionality. This event reports the load status, the version, and if it is compatible with this library. If it is not compatible, then no code objects or waves will be reported to exist.

**AMD\_DBGAPI\_EVENT\_KIND\_QUEUE\_ERROR** An event has occurred that is causing the queue to enter the error state. All non-stopped waves executing on the queue will have been stopped and a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) event will proceed this event. All waves on the queue will include the [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_QUEUE\\_ERROR](#) stop reason. No further waves will be started on the queue. The [AMD\\_DBGAPI\\_QUEUE\\_INFO\\_ERROR\\_REASON](#) query will include the union of the reasons that were reported. Some waves may be stopped before they were able to report a queue error condition. The wave stop reason will only include the reasons that were reported.

For example, if many waves encounter a memory violation at the same time, only some of the waves may report it before all the waves in the queue are stopped. Only the waves that were able to report the memory violation before all the waves were stopped will include the [AMD\\_DBGAPI\\_WAVE\\_STOP\\_REASON\\_MEMORY\\_VIOLATION](#) stop reason.

The queue error will not be reported to the inferior runtime until this event is reported as complete by calling [amd\\_dbgapi\\_event\\_processed](#). Once reported to the inferior runtime, it may cause the application to be notified which may delete and re-create the queue in order to continue submitting dispatches to the AMD GPU. If the application deletes a queue then all information about the waves executing on the queue will be lost, preventing the user from determining if a wave caused the error.

Therefore, the client may choose to stop inferior threads before reporting the event as complete. This would prevent the queue error from causing the queue to be deleted, allowing the user to inspect all the waves in the queue. Alternatively, the client may not report the event as complete until the user explicitly requests the queue error to be passed on to the inferior runtime.

### 2.16.3.3 enum amd\_dbgapi\_runtime\_state\_t

Inferior runtime state.

Enumerator

**AMD\_DBGAPI\_RUNTIME\_STATE\_LOADED\_SUPPORTED** The runtime has been loaded and is supported by the library.

**AMD\_DBGAPI\_RUNTIME\_STATE\_LOADED\_UNSUPPORTED** The runtime has been loaded but is not supported by the library.

**AMD\_DBGAPI\_RUNTIME\_STATE\_UNLOADED** The runtime has been unloaded.

## 2.16.4 Function Documentation

2.16.4.1 **amd\_dbgapi\_status\_t** **AMD\_DBGAPI** **amd\_dbgapi\_event\_get\_info** ( **amd\_dbgapi\_process\_id\_t** *process\_id*, **amd\_dbgapi\_event\_id\_t** *event\_id*, **amd\_dbgapi\_event\_info\_t** *query*, **size\_t** *value\_size*, void \* *value* )

Query information about an event.

[amd\\_dbgapi\\_event\\_info\\_t](#) specifies the queries supported and the type returned using the *value* argument.

Parameters

in	<i>process_id</i>	The process to which <i>event_id</i> belongs.
in	<i>event_id</i>	The event being queried.
in	<i>query</i>	The query being requested.
in	<i>value_size</i>	Size of the memory pointed to by <i>value</i> . Must be equal to the byte size of the query result.
out	<i>value</i>	Pointer to memory where the query result is stored.

Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the result is stored in <i>value</i> .
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	The <i>process_id</i> is invalid. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_EVENT_ID</a>	<i>event_id</i> is invalid or the NULL event. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<i>value</i> is NULL or <i>query</i> is for an attribute not present for the kind of the event. <i>value</i> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE</a>	<i>value_size</i> does not match the size of the result. <i>value</i> is unaltered.



<a href="#">AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK</a>	This will be reported if the <a href="#">amd_dbgapi_callbacks_s::allocate_memory</a> callback used to allocate <code>value</code> returns NULL. <code>value</code> is unaltered.
---	--

2.16.4.2 `amd_dbgapi_status_t` **AMD\_DBGAPI** `amd_dbgapi_event_processed ( amd_dbgapi_process_id_t process_id, amd_dbgapi_event_id_t event_id )`

Report that an event has been processed.

Every event returned by [amd\\_dbgapi\\_next\\_pending\\_event](#) must be reported as processed exactly once.

#### Parameters

in	<i>process_id</i>	The process to which <code>event_id</code> belongs.
in	<i>event_id</i>	The event that has been processed.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and the event has been reported as processed. The <code>event_id</code> is invalidated.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	The <code>process_id</code> is invalid. No event is marked as processed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_EVENT_ID</a>	The <code>event_id</code> is invalid or the NULL event. No event is marked as processed.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>event_id</code> or <code>kind</code> are NULL. No event is marked as processed.

2.16.4.3 `amd_dbgapi_status_t` **AMD\_DBGAPI** `amd_dbgapi_next_pending_event ( amd_dbgapi_process_id_t process_id, amd_dbgapi_event_id_t* event_id, amd_dbgapi_event_kind_t* kind )`

Obtain the next pending event for a process.

#### Parameters

in	<i>process_id</i>	The process from which to retrieve pending events.
out	<i>event_id</i>	The event handle of the next pending event. Each event is only returned once. If there are no pending events the <a href="#">AMD_DBGAPI_EVENT_NONE</a> handle is returned.
out	<i>kind</i>	The kind of the returned event. If there are no pending events, then <a href="#">AMD_DBGAPI_EVENT_KIND_NONE</a> is returned.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and an event or the NULL event has been returned.
---	---

<i>AMD_DBGAPI_STATUS_FATAL</i>	A fatal error occurred. The library is left uninitialized and <code>event_id</code> and <code>kind</code> are unaltered.
<i>AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</i>	The library is not initialized. The library is left uninitialized and <code>event_id</code> and <code>kind</code> are unaltered.
<i>AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</i>	The <code>process_id</code> is invalid. No event is retrieved and <code>event_id</code> and <code>kind</code> are unaltered.
<i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</i>	<code>event_id</code> or <code>kind</code> are NULL. No event is retrieved and <code>event_id</code> and <code>kind</code> are unaltered.

## 2.17 Logging

Control logging.

### Enumerations

- enum `amd_dbgapi_log_level_t` {  
`AMD_DBGAPI_LOG_LEVEL_NONE` = 0, `AMD_DBGAPI_LOG_LEVEL_FATAL_ERROR` = 1, `AMD_DBGAPI_LOG_LEVEL_WARNING` = 2, `AMD_DBGAPI_LOG_LEVEL_INFO` = 3,  
`AMD_DBGAPI_LOG_LEVEL_VERBOSE` = 4 }

*The logging levels supported.*

### Functions

- void `AMD_DBGAPI amd_dbgapi_set_log_level` (`amd_dbgapi_log_level_t` level)

*Set the logging level.*

#### 2.17.1 Detailed Description

Control logging. When the library is initially loaded the logging level is set to `AMD_DBGAPI_LOG_LEVEL_NONE`. The log level is not changed by `amd_dbgapi_initialize` or `amd_dbgapi_finalize`.

The log messages are delivered to the client using the `amd_dbgapi_callbacks_s::log_message` call back.

Note that logging can be helpful for debugging.

#### 2.17.2 Enumeration Type Documentation

##### 2.17.2.1 enum `amd_dbgapi_log_level_t`

The logging levels supported.

##### Enumerator

**`AMD_DBGAPI_LOG_LEVEL_NONE`** Print no messages.

**`AMD_DBGAPI_LOG_LEVEL_FATAL_ERROR`** Print fatal error messages. Any library function that returns the `AMD_DBGAPI_STATUS_FATAL` status code also logs a message with this level.

**`AMD_DBGAPI_LOG_LEVEL_WARNING`** Print fatal error and warning messages.

**`AMD_DBGAPI_LOG_LEVEL_INFO`** Print fatal error, warning, and info messages.

**`AMD_DBGAPI_LOG_LEVEL_VERBOSE`** Print fatal error, warning, info, and verbose messages.

#### 2.17.3 Function Documentation

##### 2.17.3.1 void `AMD_DBGAPI amd_dbgapi_set_log_level` ( `amd_dbgapi_log_level_t` level )

Set the logging level.

Internal logging messages less than the set logging level will not be reported. If `AMD_DBGAPI_LOG_LEVEL_NONE` then no messages will be reported.

This function can be used even when the library is uninitialized. However, no messages will be reported until the library is initialized when the callbacks are provided.

## Parameters

<i>in</i>	<i>level</i>	The logging level to set.
-----------	--------------	---------------------------

## Return values

<a href="#"><i>AMD_DBGAPI_STATUS_SUCCESS</i></a>	The function has been executed successfully.
<a href="#"><i>AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</i></a>	<code>level</code> is invalid. The logging level is not changed.

## 2.18 Callbacks

The library requires the client to provide a number of services.

### Data Structures

- struct [amd\\_dbgapi\\_shared\\_library\\_id\\_t](#)  
*Opaque shared library handle.*
- struct [amd\\_dbgapi\\_breakpoint\\_id\\_t](#)  
*Opaque breakpoint handle.*
- struct [amd\\_dbgapi\\_callbacks\\_s](#)  
*Callbacks that the client of the library must provide.*

### Macros

- #define [AMD\\_DBGAPI\\_SHARED\\_LIBRARY\\_NONE](#) ([amd\\_dbgapi\\_shared\\_library\\_id\\_t](#){ 0 })  
*The NULL shared library handle.*
- #define [AMD\\_DBGAPI\\_BREAKPOINT\\_NONE](#) ([amd\\_dbgapi\\_breakpoint\\_id\\_t](#){ 0 })  
*The NULL breakpoint handle.*

### Typedefs

- typedef struct  
[amd\\_dbgapi\\_callbacks\\_s](#) [amd\\_dbgapi\\_callbacks\\_t](#)  
*Forward declaration of callbacks used to specify services that must be provided by the client.*
- typedef void \* [amd\\_dbgapi\\_client\\_thread\\_id\\_t](#)  
*Opaque client thread handle.*

### Enumerations

- enum [amd\\_dbgapi\\_shared\\_library\\_state\\_t](#) { [AMD\\_DBGAPI\\_SHARED\\_LIBRARY\\_STATE\\_LOADED](#) = 1, [AMD\\_DBGAPI\\_SHARED\\_LIBRARY\\_STATE\\_UNLOADED](#) = 2 }  
*The state of a shared library.*
- enum [amd\\_dbgapi\\_breakpoint\\_action\\_t](#) { [AMD\\_DBGAPI\\_BREAKPOINT\\_ACTION\\_RESUME](#) = 1, [AMD\\_DBGAPI\\_BREAKPOINT\\_ACTION\\_HALT](#) = 2 }  
*The action to perform after reporting a breakpoint has been hit.*
- enum [amd\\_dbgapi\\_breakpoint\\_state\\_t](#) { [AMD\\_DBGAPI\\_BREAKPOINT\\_STATE\\_DISABLE](#) = 1, [AMD\\_DBGAPI\\_BREAKPOINT\\_STATE\\_ENABLE](#) = 2 }  
*The state of a breakpoint.*

### Functions

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_report\\_shared\\_library](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_shared\\_library\\_id\\_t](#) shared\_library\_id, [amd\\_dbgapi\\_shared\\_library\\_state\\_t](#) shared\_library\_state)  
*Report that a shared library enabled by the [amd\\_dbgapi\\_callbacks\\_s::enable\\_notify\\_shared\\_library](#) callback has been loaded or unloaded.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_report_breakpoint_hit (amd_dbgapi_process_id_t process_id, amd_dbgapi_breakpoint_id_t breakpoint_id, amd_dbgapi_client_thread_id_t client_thread_id, amd_dbgapi_breakpoint_action_t *breakpoint_action)`

*Report that a breakpoint added by the `amd_dbgapi_callbacks_s::add_breakpoint` callback has been hit.*

### 2.18.1 Detailed Description

The library requires the client to provide a number of services. These services are specified by providing callbacks when initializing the library using `amd_dbgapi_initialize`.

The callbacks defined in this section are invoked by the library and must not themselves invoke any function provided by the library before returning.

### 2.18.2 Macro Definition Documentation

2.18.2.1 `#define AMD_DBGAPI_BREAKPOINT_NONE (amd_dbgapi_breakpoint_id_t{0})`

The NULL breakpoint handle.

2.18.2.2 `#define AMD_DBGAPI_SHARED_LIBRARY_NONE (amd_dbgapi_shared_library_id_t{0})`

The NULL shared library handle.

### 2.18.3 Typedef Documentation

2.18.3.1 `typedef struct amd_dbgapi_callbacks_s amd_dbgapi_callbacks_t`

Forward declaration of callbacks used to specify services that must be provided by the client.

2.18.3.2 `typedef void* amd_dbgapi_client_thread_id_t`

Opaque client thread handle.

A pointer to client data associated with a thread. This pointer is passed in to the `amd_dbgapi_report_breakpoint_hit` so it can be passed out by the `AMD_DBGAPI_EVENT_KIND_BREAKPOINT_RESUME` event to allow the client of the library to identify the thread that must be resumed.

### 2.18.4 Enumeration Type Documentation

2.18.4.1 `enum amd_dbgapi_breakpoint_action_t`

The action to perform after reporting a breakpoint has been hit.

Enumerator

**`AMD_DBGAPI_BREAKPOINT_ACTION_RESUME`** Resume execution.

**`AMD_DBGAPI_BREAKPOINT_ACTION_HALT`** Leave execution halted.

### 2.18.4.2 enum amd\_dbgapi\_breakpoint\_state\_t

The state of a breakpoint.

Enumerator

**AMD\_DBGAPI\_BREAKPOINT\_STATE\_DISABLE** Breakpoint is disabled and will not report breakpoint hits.

**AMD\_DBGAPI\_BREAKPOINT\_STATE\_ENABLE** Breakpoint is enabled and will report breakpoint hits.

### 2.18.4.3 enum amd\_dbgapi\_shared\_library\_state\_t

The state of a shared library.

Enumerator

**AMD\_DBGAPI\_SHARED\_LIBRARY\_STATE\_LOADED** The shared library is loaded.

**AMD\_DBGAPI\_SHARED\_LIBRARY\_STATE\_UNLOADED** The shared library is unloaded.

## 2.18.5 Function Documentation

**2.18.5.1** `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_report_breakpoint_hit ( amd_dbgapi_process_id_t process_id, amd_dbgapi_breakpoint_id_t breakpoint_id, amd_dbgapi_client_thread_id_t client_thread_id, amd_dbgapi_breakpoint_action_t * breakpoint_action )`

Report that a breakpoint added by the [amd\\_dbgapi\\_callbacks\\_s::add\\_breakpoint](#) callback has been hit.

The thread that hit the breakpoint must remain halted while this function executes, at which point it must be resumed if `breakpoint_action` is [AMD\\_DBGAPI\\_BREAKPOINT\\_ACTION\\_RESUME](#). If `breakpoint_action` is `:AMD_DBGAPI_BREAKPOINT_ACTION_HALT` then the client should process pending events which will cause a [AMD\\_DBGAPI\\_EVENT\\_KIND\\_BREAKPOINT\\_RESUME](#) event to be added which specifies that the thread should now be resumed.

Parameters

in	<i>process_id</i>	The process to which the <code>client_thread_id</code> hitting the breakpoint belongs.
in	<i>breakpoint_id</i>	The breakpoint that has been hit.
in	<i>client_thread_id</i>	The client identification of the thread that hit the breakpoint.
out	<i>breakpoint_action</i>	Indicate if the thread hitting the breakpoint should be resumed or remain halted when this function returns.

Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully and <code>breakpoint_action</code> indicates if the thread hitting the breakpoint should be resumed.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The library is left uninitialized and <code>breakpoint_action</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The library is not initialized. The library is left uninitialized and <code>breakpoint_action</code> is unaltered.



<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	The <code>process_id</code> is invalid. <code>breakpoint_action</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_BREAKPOINT_ID</a>	The <code>breakpoint_id</code> is invalid. <code>breakpoint_action</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>breakpoint_action</code> is NULL. <code>breakpoint_action</code> is unaltered.

2.18.5.2 `amd_dbgapi_status_t` `AMD_DBGAPI amd_dbgapi_report_shared_library ( amd_dbgapi_process_id_t process_id, amd_dbgapi_shared_library_id_t shared_library_id, amd_dbgapi_shared_library_state_t shared_library_state )`

Report that a shared library enabled by the [amd\\_dbgapi\\_callbacks\\_s::enable\\_notify\\_shared\\_library](#) callback has been loaded or unloaded.

The thread that is performing the shared library load or unload must remain halted while this function executes. This allows the library to use the [amd\\_dbgapi\\_callbacks\\_s::get\\_symbol\\_address](#), [amd\\_dbgapi\\_callbacks\\_s::add\\_breakpoint](#) and [amd\\_dbgapi\\_callbacks\\_s::remove\\_breakpoint](#) callbacks to add or remove breakpoints on library load or unload respectively. The breakpoints must be added before any code can execute in the shared library.

#### Parameters

in	<code>process_id</code>	The process to which the <code>shared_library_id</code> belongs.
in	<code>shared_library_id</code>	The shared library that has been loaded or unloaded.
in	<code>shared_library_state</code>	The shared library state.

#### Return values

<a href="#">AMD_DBGAPI_STATUS_SUCCESS</a>	The function has been executed successfully.
<a href="#">AMD_DBGAPI_STATUS_FATAL</a>	A fatal error occurred. The amd-dbgapi library is left uninitialized and <code>resume</code> is unaltered.
<a href="#">AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED</a>	The amd-dbgapi library is not initialized. The amd-dbgapi library is left uninitialized.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID</a>	The <code>process_id</code> is invalid.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_SHARED_LIBRARY_ID</a>	The <code>shared_library_id</code> is invalid.
<a href="#">AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT</a>	<code>shared_library_state</code> is invalid.
<a href="#">AMD_DBGAPI_STATUS_ERROR</a>	<code>shared_library_state</code> is not consistent with the previously reported load state. For example, it is reported as loaded when previously also reported as loaded.



## Chapter 3

# Data Structure Documentation

### 3.1 amd\_dbgapi\_address\_class\_id\_t Struct Reference

Opaque source language address class handle.

```
#include <amd_dbgapi.h>
```

#### Data Fields

- uint64\_t [handle](#)

#### 3.1.1 Detailed Description

Opaque source language address class handle.

A source language address class describes the source language address spaces. It is used to define source language pointer and reference types. Each architecture has its own mapping of them to the architecture specific address spaces.

The handle is only unique within a specific architecture.

See [User Guide for AMDGPU Backend – Code Object – DWARF – Address Class Mapping](#).

#### 3.1.2 Field Documentation

##### 3.1.2.1 uint64\_t amd\_dbgapi\_address\_class\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd\\_dbgapi.h](#)

### 3.2 amd\_dbgapi\_address\_space\_id\_t Struct Reference

Opaque address space handle.

```
#include <amd_dbgapi.h>
```

## Data Fields

- uint64\_t [handle](#)

### 3.2.1 Detailed Description

Opaque address space handle.

A handle that denotes the set of address spaces supported by an architecture.

The handle is only unique within a specific architecture.

See [User Guide for AMDGPU Backend - LLVM - Address Spaces](#).

### 3.2.2 Field Documentation

#### 3.2.2.1 uint64\_t amd\_dbgapi\_address\_space\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.3 amd\_dbgapi\_agent\_id\_t Struct Reference

Opaque agent handle.

```
#include <amd-dbgapi.h>
```

## Data Fields

- uint64\_t [handle](#)

### 3.3.1 Detailed Description

Opaque agent handle.

Only unique within a single process.

### 3.3.2 Field Documentation

#### 3.3.2.1 uint64\_t amd\_dbgapi\_agent\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.4 amd\_dbgapi\_architecture\_id\_t Struct Reference

Opaque architecture handle.

```
#include <amd_dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

#### 3.4.1 Detailed Description

Opaque architecture handle.

An architecture handle is unique for each AMD GPU model supported by the library. They are only valid while the library is initialized and are invalidated when the library is uninitialized.

#### 3.4.2 Field Documentation

##### 3.4.2.1 uint64\_t amd\_dbgapi\_architecture\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd\\_dbgapi.h](#)

## 3.5 amd\_dbgapi\_breakpoint\_id\_t Struct Reference

Opaque breakpoint handle.

```
#include <amd_dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

#### 3.5.1 Detailed Description

Opaque breakpoint handle.

Only unique within a single process.

The implementation of the library requests the client to set breakpoints in certain functions so that it can be notified when certain actions are being performed, and to stop the thread performing the action. This allows the data to be retrieved and updated without conflicting with the thread. The library will resume the thread when it has completed the access.

#### 3.5.2 Field Documentation

##### 3.5.2.1 uint64\_t amd\_dbgapi\_breakpoint\_id\_t::handle

The documentation for this struct was generated from the following file:

- `include/amd-dbgapi.h`

## 3.6 amd\_dbgapi\_callbacks\_s Struct Reference

Callbacks that the client of the library must provide.

```
#include <amd-dbgapi.h>
```

### Data Fields

- `void *(* allocate_memory )(size_t byte_size)`  
*Allocate memory to be used to return a value from the library that is then owned by the client.*
- `void(* deallocate_memory )(void *data)`  
*Deallocate memory that was allocated by `amd_dbgapi_callbacks_s::allocate_memory`.*
- `amd_dbgapi_status_t(* get_os_pid )(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_os_pid *os_pid)`  
*Return the native operating system process handle for the process identified by the client process handle.*
- `amd_dbgapi_status_t(* enable_notify_shared_library )(amd_dbgapi_client_process_id_t client_process_id, const char *shared_library_name, amd_dbgapi_shared_library_id_t shared_library_id, amd_dbgapi_shared_library_state_t *shared_library_state)`  
*Request to be notified when a shared library is loaded and unloaded.*
- `amd_dbgapi_status_t(* disable_notify_shared_library )(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_shared_library_id_t shared_library_id)`  
*Request to stop being notified for a shared library previously set by `amd_dbgapi_callbacks_s::enable_notify_shared_library`.*
- `amd_dbgapi_status_t(* get_symbol_address )(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_shared_library_id_t shared_library_id, const char *symbol_name, amd_dbgapi_global_address_t *address)`  
*Return the address of a symbol in a shared library.*
- `amd_dbgapi_status_t(* add_breakpoint )(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_shared_library_id_t shared_library_id, amd_dbgapi_global_address_t address, amd_dbgapi_breakpoint_id_t breakpoint_id)`  
*Add a breakpoint in a shared library using a global address.*
- `amd_dbgapi_status_t(* remove_breakpoint )(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_breakpoint_id_t breakpoint_id)`  
*Remove a breakpoint previously added by `amd_dbgapi_callbacks_s::add_breakpoint`.*
- `amd_dbgapi_status_t(* set_breakpoint_state )(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_breakpoint_id_t breakpoint_id, amd_dbgapi_breakpoint_state_t breakpoint_state)`  
*Set the state of a breakpoint previously added by `amd_dbgapi_callbacks_s::add_breakpoint`.*
- `void(* log_message )(amd_dbgapi_log_level_t level, const char *message)`  
*Report a log message.*

### 3.6.1 Detailed Description

Callbacks that the client of the library must provide.

The client implementation of the callbacks must not invoke any operation of the library.

### 3.6.2 Field Documentation

**3.6.2.1** `amd_dbgapi_status_t(* amd_dbgapi_callbacks_s::add_breakpoint)(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_shared_library_id_t shared_library_id, amd_dbgapi_global_address_t address, amd_dbgapi_breakpoint_id_t breakpoint_id)`

Add a breakpoint in a shared library using a global address.

The library only adds breakpoints in loaded shared libraries, will request to be notified when the shared library is unloaded, and remove them when notified that the shared library is unloaded.

Breakpoints must be added in the [AMD\\_DBGAPI\\_BREAKPOINT\\_STATE\\_ENABLE](#) state.

`client_process_id` is the client handle of the process in which the breakpoint is to be added.

`shared_library_id` is the shared library that contains the address.

`address` is the global address to add the breakpoint.

`breakpoint_id` is the handle to identify this breakpoint which must be specified when [amd\\_dbgapi\\_report\\_breakpoint\\_hit](#) is used to report a breakpoint hit, and in the [AMD\\_DBGAPI\\_EVENT\\_KIND\\_BREAKPOINT\\_RESUME](#) event that may be used to resume the thread.

Return [AMD\\_DBGAPI\\_STATUS\\_SUCCESS](#) if successful. The breakpoint is added.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_CLIENT\\_PROCESS\\_ID](#) if the `client_process_id` handle is invalid. No breakpoint is added.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_SHARED\\_LIBRARY\\_ID](#) if the `shared_library` handle is invalid. No breakpoint is added.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_LIBRARY\\_NOT\\_LOADED](#) if `code_object_name` shared library is not currently loaded. No breakpoint is added.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_ADDRESS](#) if `address` is not an address in shared library `shared_library_id`. No breakpoint is added.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_BREAKPOINT\\_ID](#) if there is a breakpoint already added with `breakpoint_id`. No breakpoint is added.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR](#) if another error was encountered. No breakpoint is added.

**3.6.2.2** `void(* amd_dbgapi_callbacks_s::allocate_memory)(size_t byte_size)`

Allocate memory to be used to return a value from the library that is then owned by the client.

The memory should be suitably aligned for any type. If unable to allocate memory of the byte size specified by `byte_size` then return NULL. The client is responsible for deallocating this memory, and so is responsible for tracking the size of the allocation. Note that these requirements can be met by implementing using `malloc`.

**3.6.2.3** `void(* amd_dbgapi_callbacks_s::deallocate_memory)(void *data)`

Deallocate memory that was allocated by [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#).

`data` is a pointer returned by [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) that will not be returned to the client and has not been previously deallocated. Note this callback may be used by the library implementation if it encounters an error after using [amd\\_dbgapi\\_callbacks\\_s::allocate\\_memory](#) to allocate memory.

### 3.6.2.4 `amd_dbgapi_status_t(*amd_dbgapi_callbacks_s::disable_notify_shared_library)(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_shared_library_id_t shared_library_id)`

Request to stop being notified for a shared library previously set by `amd_dbgapi_callbacks_s::enable_notify_shared_library`.

`shared_library_id` is invalidated.

`client_process_id` is the client handle of the process in which loading of the shared library is being notified.

`shared_library_id` is the handle of the shared library to stop being notified.

Return `AMD_DBGAPI_STATUS_SUCCESS` if successful.

Return `AMD_DBGAPI_STATUS_ERROR_INVALID_CLIENT_PROCESS_ID` if the `client_process_id` handle is invalid.

Return `AMD_DBGAPI_STATUS_ERROR_INVALID_SHARED_LIBRARY_ID` if the `shared_library` handle is invalid.

Return `AMD_DBGAPI_STATUS_ERROR` if an error was encountered.

### 3.6.2.5 `amd_dbgapi_status_t(*amd_dbgapi_callbacks_s::enable_notify_shared_library)(amd_dbgapi_client_process_id_t client_process_id, const char *shared_library_name, amd_dbgapi_shared_library_id_t shared_library_id, amd_dbgapi_shared_library_state_t *shared_library_state)`

Request to be notified when a shared library is loaded and unloaded.

If multiple shared libraries match the name, then the client must only associate `shared_library_id` with a single shared library, and only invoke `amd_dbgapi_report_shared_library` for that single shared library.

`client_process_id` is the client handle of the process in which loading of the shared library must be notified.

`shared_library_name` is the name of the shared library being requested. The name is a path of the shared library and can contain the `*` character which matches any characters. The memory is owned by the library and is only valid while the callback executes.

`shared_library_id` is the handle to identify this shared library which must be specified when `amd_dbgapi_report_shared_library` is used to report a shared library load or unload.

`shared_library_state` must be set to a value that indicates whether the shared library is already loaded.

Return `AMD_DBGAPI_STATUS_SUCCESS` if successful.

Return `AMD_DBGAPI_STATUS_ERROR_INVALID_CLIENT_PROCESS_ID` if the `client_process_id` handle is invalid.

Return `AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT` if `shared_library_name` or `shared_library_state` are NULL or `shared_library_name` is an invalid library name.

Return `AMD_DBGAPI_STATUS_ERROR` if another error was encountered.

### 3.6.2.6 `amd_dbgapi_status_t(*amd_dbgapi_callbacks_s::get_os_pid)(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_os_pid_t *os_pid)`

Return the native operating system process handle for the process identified by the client process handle.

This value is required to not change during the lifetime of the process associated with the client process handle.

For Linux<sup>®</sup> this is the `pid_t` from `sys/types.h` and is required to have already been `ptrace` enabled.

`client_process_id` is the client handle of the process for which the operating system process handle is being



queried.

`os_pid` must be set to the native operating system process handle.

Return [AMD\\_DBGAPI\\_STATUS\\_SUCCESS](#) if successful and `os_pid` is updated.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_CLIENT\\_PROCESS\\_ID](#) if the `client_process_id` handle is invalid.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_PROCESS\\_EXITED](#) if the `client_process_id` handle is associated with a native operating system process that has already exited.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_ARGUMENT](#) if `os_pid` is NULL.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR](#) if an error was encountered.

**3.6.2.7** `amd_dbgapi_status_t(* amd_dbgapi_callbacks_s::get_symbol_address)(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_shared_library_id_t shared_library_id, const char *symbol_name, amd_dbgapi_global_address_t *address)`

Return the address of a symbol in a shared library.

`client_process_id` is the client handle of the process being queried.

`shared_library_id` is the shared library that contains the symbol.

`symbol_name` is the name of the symbol being requested. The memory is owned by the library and is only valid while the callback executes.

`address` must be updated with the address of the symbol.

Return [AMD\\_DBGAPI\\_STATUS\\_SUCCESS](#) if successful.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_CLIENT\\_PROCESS\\_ID](#) if the `client_process_id` handle is invalid.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_SHARED\\_LIBRARY\\_ID](#) if the `shared_library` handle is invalid.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_LIBRARY\\_NOT\\_LOADED](#) if `code_object_name` shared library is not currently loaded.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_SYMBOL\\_NOT\\_FOUND](#) if `shared_library_id` shared library is loaded but does not contain `symbol_name`.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_ARGUMENT](#) if `symbol_name` or `address` are NULL.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR](#) if an error was encountered.

**3.6.2.8** `void(* amd_dbgapi_callbacks_s::log_message)(amd_dbgapi_log_level_t level, const char *message)`

Report a log message.

`level` is the log level.

`message` is a NUL terminated string to print that is owned by the library and is only valid while the callback executes.

**3.6.2.9** `amd_dbgapi_status_t(* amd_dbgapi_callbacks_s::remove_breakpoint)(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_breakpoint_id_t breakpoint_id)`

Remove a breakpoint previously added by [amd\\_dbgapi\\_callbacks\\_s::add\\_breakpoint](#).

`breakpoint_id` is invalidated.

`client_process_id` is the client handle of the process in which the breakpoint is to be removed.

`breakpoint_id` is the breakpoint handle of the breakpoint to remove.

Return [AMD\\_DBGAPI\\_STATUS\\_SUCCESS](#) if successful. The breakpoint is removed.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_CLIENT\\_PROCESS\\_ID](#) if the `client_process_id` handle is invalid. No breakpoint is removed.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_BREAKPOINT\\_ID](#) if `breakpoint_id` handle is invalid. No breakpoint is removed.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_LIBRARY\\_NOT\\_LOADED](#) if the shared library containing the breakpoint is not currently loaded. The breakpoint will already have been removed.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR](#) if another error was encountered.

**3.6.2.10** `amd_dbgapi_status_t(*amd_dbgapi_callbacks_s::set_breakpoint_state)(amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_breakpoint_id_t breakpoint_id, amd_dbgapi_breakpoint_state_t breakpoint_state)`

Set the state of a breakpoint previously added by [amd\\_dbgapi\\_callbacks\\_s::add\\_breakpoint](#).

`client_process_id` is the client handle of the process in which the breakpoint is added.

`breakpoint_id` is the breakpoint handle of the breakpoint to update.

`breakpoint_state` is the state to which to set the breakpoint.

Return [AMD\\_DBGAPI\\_STATUS\\_SUCCESS](#) if successful. The breakpoint is set to the requested state.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_CLIENT\\_PROCESS\\_ID](#) if the `client_process_id` handle is invalid. No breakpoint is update.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_BREAKPOINT\\_ID](#) if `breakpoint_id` handle is invalid. No breakpoint is updated.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_LIBRARY\\_NOT\\_LOADED](#) if the shared library containing the breakpoint is not currently loaded. The breakpoint will have been removed. `breakpoint_id` is invalidated.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR\\_INVALID\\_ARGUMENT](#) if `breakpoint_state` is invalid.

Return [AMD\\_DBGAPI\\_STATUS\\_ERROR](#) if another error was encountered.

The documentation for this struct was generated from the following file:

- [include/amd-dbgapi.h](#)

## 3.7 `amd_dbgapi_code_object_id_t` Struct Reference

Opaque code object handle.

```
#include <amd-dbgapi.h>
```

### Data Fields

- `uint64_t` [handle](#)

### 3.7.1 Detailed Description

Opaque code object handle.

Only unique within a single process.

### 3.7.2 Field Documentation

#### 3.7.2.1 uint64\_t amd\_dbgapi\_code\_object\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.8 amd\_dbgapi\_dispatch\_id\_t Struct Reference

Opaque dispatch handle.

```
#include <amd-dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

### 3.8.1 Detailed Description

Opaque dispatch handle.

Only unique within a single process.

### 3.8.2 Field Documentation

#### 3.8.2.1 uint64\_t amd\_dbgapi\_dispatch\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.9 amd\_dbgapi\_displaced\_stepping\_id\_t Struct Reference

Opaque displaced stepping handle.

```
#include <amd-dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

### 3.9.1 Detailed Description

Opaque displaced stepping handle.

Only unique within a single process.

### 3.9.2 Field Documentation

#### 3.9.2.1 uint64\_t amd\_dbgapi\_displaced\_stepping\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.10 amd\_dbgapi\_event\_id\_t Struct Reference

Opaque event handle.

```
#include <amd-dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

### 3.10.1 Detailed Description

Opaque event handle.

Only unique within a single process.

### 3.10.2 Field Documentation

#### 3.10.2.1 uint64\_t amd\_dbgapi\_event\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.11 amd\_dbgapi\_process\_id\_t Struct Reference

Opaque process handle.

```
#include <amd-dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

### 3.11.1 Detailed Description

Opaque process handle.

Unique for a single library initialization.

All operations that control an AMD GPU specify the process that is using the AMD GPU with the process handle. It is undefined to use handles returned by operations performed for one process, with operations performed for a different process.

### 3.11.2 Field Documentation

#### 3.11.2.1 uint64\_t amd\_dbgapi\_process\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.12 amd\_dbgapi\_queue\_id\_t Struct Reference

Opaque queue handle.

```
#include <amd-dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

### 3.12.1 Detailed Description

Opaque queue handle.

Only unique within a single process.

### 3.12.2 Field Documentation

#### 3.12.2.1 uint64\_t amd\_dbgapi\_queue\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.13 amd\_dbgapi\_register\_class\_id\_t Struct Reference

Opaque register class handle.

```
#include <amd-dbgapi.h>
```

## Data Fields

- uint64\_t [handle](#)

### 3.13.1 Detailed Description

Opaque register class handle.

A handle that denotes the set of classes of hardware registers supported by an architecture. The registers of the architecture all belong to one or more register classes. The register classes are a convenience for grouping registers that have similar uses and properties. They can be useful when presenting register lists to a user. For example, there could be a register class for *system*, *general*, and *vector*.

The handle is only unique within a specific architecture.

### 3.13.2 Field Documentation

#### 3.13.2.1 uint64\_t amd\_dbgapi\_register\_class\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.14 amd\_dbgapi\_register\_id\_t Struct Reference

Opaque register handle.

```
#include <amd-dbgapi.h>
```

## Data Fields

- uint64\_t [handle](#)

### 3.14.1 Detailed Description

Opaque register handle.

A handle that denotes the set of hardware registers supported by an architecture.

The handle is only unique within a specific architecture.

### 3.14.2 Field Documentation

#### 3.14.2.1 uint64\_t amd\_dbgapi\_register\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd-dbgapi.h](#)

## 3.15 amd\_dbgapi\_shared\_library\_id\_t Struct Reference

Opaque shared library handle.

```
#include <amd_dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

#### 3.15.1 Detailed Description

Opaque shared library handle.

Only unique within a single process.

The implementation of the library requests the client to notify it when a specific shared library is loaded and unloaded. This allows the library to set breakpoints within the shared library and access global variable data within it.

#### 3.15.2 Field Documentation

##### 3.15.2.1 uint64\_t amd\_dbgapi\_shared\_library\_id\_t::handle

The documentation for this struct was generated from the following file:

- include/[amd\\_dbgapi.h](#)

## 3.16 amd\_dbgapi\_wave\_id\_t Struct Reference

Opaque wave handle.

```
#include <amd_dbgapi.h>
```

### Data Fields

- uint64\_t [handle](#)

#### 3.16.1 Detailed Description

Opaque wave handle.

Waves are the way the AMD GPU executes code.

Only unique within a single process.

#### 3.16.2 Field Documentation

##### 3.16.2.1 uint64\_t amd\_dbgapi\_wave\_id\_t::handle

The documentation for this struct was generated from the following file:

- [include/amd-dbgapi.h](#)



## Chapter 4

# File Documentation

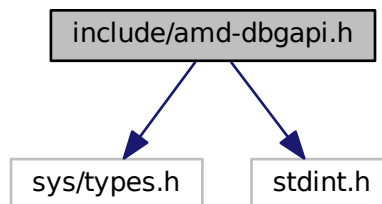
### 4.1 include/amd-dbgapi.h File Reference

AMD debugger API interface.

```
#include <sys/types.h>
```

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

Include dependency graph for amd-dbgapi.h:



### Data Structures

- struct [amd\\_dbgapi\\_architecture\\_id\\_t](#)  
*Opaque architecture handle.*
- struct [amd\\_dbgapi\\_process\\_id\\_t](#)  
*Opaque process handle.*
- struct [amd\\_dbgapi\\_code\\_object\\_id\\_t](#)  
*Opaque code object handle.*
- struct [amd\\_dbgapi\\_agent\\_id\\_t](#)  
*Opaque agent handle.*
- struct [amd\\_dbgapi\\_queue\\_id\\_t](#)  
*Opaque queue handle.*

- struct [amd\\_dbgapi\\_dispatch\\_id\\_t](#)  
*Opaque dispatch handle.*
- struct [amd\\_dbgapi\\_wave\\_id\\_t](#)  
*Opaque wave handle.*
- struct [amd\\_dbgapi\\_displaced\\_stepping\\_id\\_t](#)  
*Opaque displaced stepping handle.*
- struct [amd\\_dbgapi\\_register\\_class\\_id\\_t](#)  
*Opaque register class handle.*
- struct [amd\\_dbgapi\\_register\\_id\\_t](#)  
*Opaque register handle.*
- struct [amd\\_dbgapi\\_address\\_class\\_id\\_t](#)  
*Opaque source language address class handle.*
- struct [amd\\_dbgapi\\_address\\_space\\_id\\_t](#)  
*Opaque address space handle.*
- struct [amd\\_dbgapi\\_event\\_id\\_t](#)  
*Opaque event handle.*
- struct [amd\\_dbgapi\\_shared\\_library\\_id\\_t](#)  
*Opaque shared library handle.*
- struct [amd\\_dbgapi\\_breakpoint\\_id\\_t](#)  
*Opaque breakpoint handle.*
- struct [amd\\_dbgapi\\_callbacks\\_s](#)  
*Callbacks that the client of the library must provide.*

## Macros

- #define [AMD\\_DBGAPI\\_CALL](#)
- #define [AMD\\_DBGAPI\\_EXPORT](#) AMD\_DBGAPI\_EXPORT\_DECORATOR [AMD\\_DBGAPI\\_CALL](#)
- #define [AMD\\_DBGAPI\\_IMPORT](#) AMD\_DBGAPI\_IMPORT\_DECORATOR [AMD\\_DBGAPI\\_CALL](#)
- #define [AMD\\_DBGAPI\\_AMD\\_DBGAPI\\_IMPORT](#)
- #define [AMD\\_DBGAPI\\_SYMBOL\\_VERSION](#) "\_0.1"  
*The name used for the shared library versioned symbols.*
- #define [AMD\\_DBGAPI\\_ARCHITECTURE\\_NONE](#) (amd\_dbgapi\_architecture\_id\_t{ 0 })  
*The NULL architecture handle.*
- #define [AMD\\_DBGAPI\\_PROCESS\\_NONE](#) (amd\_dbgapi\_process\_id\_t{ 0 })  
*The NULL process handle.*
- #define [AMD\\_DBGAPI\\_CODE\\_OBJECT\\_NONE](#) (amd\_dbgapi\_code\_object\_id\_t{ 0 })  
*The NULL code object handle.*
- #define [AMD\\_DBGAPI\\_AGENT\\_NONE](#) (amd\_dbgapi\_agent\_id\_t{ 0 })  
*The NULL agent handle.*
- #define [AMD\\_DBGAPI\\_QUEUE\\_NONE](#) (amd\_dbgapi\_queue\_id\_t{ 0 })  
*The NULL queue handle.*
- #define [AMD\\_DBGAPI\\_DISPATCH\\_NONE](#) (amd\_dbgapi\_dispatch\_id\_t{ 0 })  
*The NULL dispatch handle.*
- #define [AMD\\_DBGAPI\\_WAVE\\_NONE](#) (amd\_dbgapi\_wave\_id\_t{ 0 })  
*The NULL wave handle.*
- #define [AMD\\_DBGAPI\\_DISPLACED\\_STEPPING\\_NONE](#) (amd\_dbgapi\_displaced\_stepping\_id\_t{ 0 })  
*The NULL displaced stepping handle.*

- #define [AMD\\_DBGAPI\\_WATCHPOINT\\_NONE](#) ([amd\\_dbgapi\\_watchpoint\\_id\\_t](#) (-1))  
*The NULL watchpoint handle.*
- #define [AMD\\_DBGAPI\\_REGISTER\\_CLASS\\_NONE](#) ([amd\\_dbgapi\\_register\\_class\\_id\\_t](#){ 0 })  
*The NULL register class handle.*
- #define [AMD\\_DBGAPI\\_REGISTER\\_NONE](#) ([amd\\_dbgapi\\_register\\_id\\_t](#){ 0 })  
*The NULL register handle.*
- #define [AMD\\_DBGAPI\\_LANE\\_NONE](#) ([amd\\_dbgapi\\_lane\\_id\\_t](#) (-1))  
*The NULL lane handle.*
- #define [AMD\\_DBGAPI\\_ADDRESS\\_CLASS\\_NONE](#) ([amd\\_dbgapi\\_address\\_class\\_id\\_t](#){ 0 })  
*The NULL address class handle.*
- #define [AMD\\_DBGAPI\\_EVENT\\_NONE](#) ([amd\\_dbgapi\\_event\\_id\\_t](#){ 0 })  
*The NULL event handle.*
- #define [AMD\\_DBGAPI\\_SHARED\\_LIBRARY\\_NONE](#) ([amd\\_dbgapi\\_shared\\_library\\_id\\_t](#){ 0 })  
*The NULL shared library handle.*
- #define [AMD\\_DBGAPI\\_BREAKPOINT\\_NONE](#) ([amd\\_dbgapi\\_breakpoint\\_id\\_t](#){ 0 })  
*The NULL breakpoint handle.*

## Typedefs

- typedef struct  
[amd\\_dbgapi\\_callbacks\\_s](#) [amd\\_dbgapi\\_callbacks\\_t](#)  
*Forward declaration of callbacks used to specify services that must be provided by the client.*
- typedef uint64\_t [amd\\_dbgapi\\_global\\_address\\_t](#)  
*Integral type used for a global virtual memory address in the inferior process.*
- typedef uint64\_t [amd\\_dbgapi\\_size\\_t](#)  
*Integral type used for sizes, including memory allocations, in the inferior.*
- typedef pid\_t [amd\\_dbgapi\\_os\\_pid](#)  
*Native operating system process id.*
- typedef int [amd\\_dbgapi\\_notifier\\_t](#)  
*Type used to notify the client of the library that a process may have pending events.*
- typedef void \* [amd\\_dbgapi\\_client\\_process\\_id\\_t](#)  
*Opaque client process handle.*
- typedef uint64\_t [amd\\_dbgapi\\_queue\\_packet\\_id\\_t](#)  
*Queue packet ID.*
- typedef uint32\_t [amd\\_dbgapi\\_watchpoint\\_id\\_t](#)  
*A hardware data watchpoint handle.*
- typedef uint32\_t [amd\\_dbgapi\\_lane\\_id\\_t](#)  
*A wave lane handle.*
- typedef uint64\_t [amd\\_dbgapi\\_segment\\_address\\_t](#)  
*Each address space has its own linear address to access it termed a segment address.*
- typedef void \* [amd\\_dbgapi\\_client\\_thread\\_id\\_t](#)  
*Opaque client thread handle.*

## Enumerations

- enum `amd_dbgapi_changed_t` { `AMD_DBGAPI_CHANGED_NO` = 0, `AMD_DBGAPI_CHANGED_YES` = 1 }

*Indication of if a value has changed.*

- enum `amd_dbgapi_status_t` {  
`AMD_DBGAPI_STATUS_SUCCESS` = 0, `AMD_DBGAPI_STATUS_ERROR` = -1, `AMD_DBGAPI_STATUS_FATAL` = -2, `AMD_DBGAPI_STATUS_ERROR_NOT_SUPPORTED` = -3,  
`AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT` = -4, `AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE` = -5, `AMD_DBGAPI_STATUS_ERROR_ALREADY_INITIALIZED` = -6, `AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED` = -7,  
`AMD_DBGAPI_STATUS_ERROR_VERSION_MISMATCH` = -8, `AMD_DBGAPI_STATUS_ERROR_ALREADY_ATTACHED` = -9, `AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID` = -10, `AMD_DBGAPI_STATUS_ERROR_ILLEGAL_INSTRUCTION` = -11,  
`AMD_DBGAPI_STATUS_ERROR_INVALID_CODE_OBJECT_ID` = -12, `AMD_DBGAPI_STATUS_ERROR_INVALID_ELF_AMDGPU_MACHINE` = -13, `AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID` = -14, `AMD_DBGAPI_STATUS_ERROR_INVALID_AGENT_ID` = -15,  
`AMD_DBGAPI_STATUS_ERROR_INVALID_QUEUE_ID` = -16, `AMD_DBGAPI_STATUS_ERROR_INVALID_DISPATCH_ID` = -17, `AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID` = -18, `AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED` = -19,  
`AMD_DBGAPI_STATUS_ERROR_WAVE_STOPPED` = -20, `AMD_DBGAPI_STATUS_ERROR_WAVE_OUTSTANDING_STOP` = -21, `AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_RESUMABLE` = -22, `AMD_DBGAPI_STATUS_ERROR_INVALID_DISPLACED_STEPPING_ID` = -23,  
`AMD_DBGAPI_STATUS_ERROR_DISPLACED_STEPPING_BUFFER_UNAVAILABLE` = -24, `AMD_DBGAPI_STATUS_ERROR_INVALID_WATCHPOINT_ID` = -25, `AMD_DBGAPI_STATUS_ERROR_NO_WATCHPOINT_AVAILABLE` = -26, `AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_CLASS_ID` = -27,  
`AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID` = -28, `AMD_DBGAPI_STATUS_ERROR_INVALID_LANE_ID` = -29, `AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_CLASS_ID` = -30, `AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID` = -31,  
`AMD_DBGAPI_STATUS_ERROR_MEMORY_ACCESS` = -32, `AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_CONVERSION` = -33, `AMD_DBGAPI_STATUS_ERROR_INVALID_EVENT_ID` = -34, `AMD_DBGAPI_STATUS_ERROR_INVALID_SHARED_LIBRARY_ID` = -35,  
`AMD_DBGAPI_STATUS_ERROR_INVALID_BREAKPOINT_ID` = -36, `AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK` = -37, `AMD_DBGAPI_STATUS_ERROR_INVALID_CLIENT_PROCESS_ID` = -38, `AMD_DBGAPI_STATUS_ERROR_PROCESS_EXITED` = -39,  
`AMD_DBGAPI_STATUS_ERROR_LIBRARY_NOT_LOADED` = -40, `AMD_DBGAPI_STATUS_ERROR_SYMBOL_NOT_FOUND` = -41, `AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS` = -42 }

*AMD debugger API status codes.*

- enum { `AMD_DBGAPI_VERSION_MAJOR` = 0, `AMD_DBGAPI_VERSION_MINOR` = 1 }

*The semantic version of the interface following [semver.org][semver] rules.*

- enum `amd_dbgapi_architecture_info_t` {  
`AMD_DBGAPI_ARCHITECTURE_INFO_NAME` = 1, `AMD_DBGAPI_ARCHITECTURE_INFO_ELF_AMDGPU_MACHINE` = 2, `AMD_DBGAPI_ARCHITECTURE_INFO_LARGEST_INSTRUCTION_SIZE` = 3, `AMD_DBGAPI_ARCHITECTURE_INFO_MINIMUM_INSTRUCTION_ALIGNMENT` = 4,  
`AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_SIZE` = 5, `AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_PC_ADJUST` = 6, `AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_PC_REGISTER` = 7, `AMD_DBGAPI_ARCHITECTURE_INFO_EXECUTION_MASK_REGISTER` = 9, `AMD_DBGAPI_ARCHITECTURE_INFO_WATCHPOINT_COUNT` = 10, `AMD_DBGAPI_ARCHITECTURE_INFO_WATCHPOINT_SHARE` = 11, `AMD_DBGAPI_ARCHITECTURE_INFO_DEFAULT_GLOBAL_ADDRESS_SPACE` = 12,  
`AMD_DBGAPI_ARCHITECTURE_INFO_PRECISE_MEMORY_SUPPORTED` = 13 }

*Architecture queries that are supported by `amd_dbgapi_architecture_get_info`.*

- enum `amd_dbgapi_process_info_t` { `AMD_DBGAPI_PROCESS_INFO_NOTIFIER` = 1 }

*Process queries that are supported by `amd_dbgapi_process_get_info`.*

- enum `amd_dbgapi_progress_t` { `AMD_DBGAPI_PROGRESS_NORMAL` = 0, `AMD_DBGAPI_PROGRESS_NO_FORWARD` = 1 }

*The kinds of progress supported by the library.*

- enum `amd_dbgapi_code_object_info_t` { `AMD_DBGAPI_CODE_OBJECT_INFO_URI_NAME` = 1, `AMD_DBGAPI_CODE_OBJECT_INFO_LOAD_ADDRESS` = 2 }

*Code object queries that are supported by `amd_dbgapi_code_object_get_info`.*

- enum `amd_dbgapi_agent_info_t` {  
`AMD_DBGAPI_AGENT_INFO_NAME` = 1, `AMD_DBGAPI_AGENT_INFO_ARCHITECTURE` = 2, `AMD_DBGAPI_AGENT_INFO_PCIE_SLOT` = 3, `AMD_DBGAPI_AGENT_INFO_PCIE_VENDOR_ID` = 4,  
`AMD_DBGAPI_AGENT_INFO_PCIE_DEVICE_ID` = 5, `AMD_DBGAPI_AGENT_INFO_SHADER_ENGINE_COUNT` = 6, `AMD_DBGAPI_AGENT_INFO_COMPUTE_UNIT_COUNT` = 7, `AMD_DBGAPI_AGENT_INFO_NUM_SIMD_PER_COMPUTE_UNIT` = 8,  
`AMD_DBGAPI_AGENT_INFO_MAX_WAVES_PER_SIMD` = 9 }

*Agent queries that are supported by `amd_dbgapi_agent_get_info`.*

- enum `amd_dbgapi_queue_info_t` {  
`AMD_DBGAPI_QUEUE_INFO_AGENT` = 1, `AMD_DBGAPI_QUEUE_INFO_ARCHITECTURE` = 2, `AMD_DBGAPI_QUEUE_TYPE` = 3, `AMD_DBGAPI_QUEUE_INFO_STATE` = 4,  
`AMD_DBGAPI_QUEUE_INFO_ERROR_REASON` = 5 }

*Queue queries that are supported by `amd_dbgapi_queue_get_info`.*

- enum `amd_dbgapi_queue_type_t` { `AMD_DBGAPI_QUEUE_TYPE_UNKNOWN` = 0, `AMD_DBGAPI_QUEUE_TYPE_HSA_KERNEL_DISPATCH_MULTIPLE_PRODUCER` = 1, `AMD_DBGAPI_QUEUE_TYPE_HSA_KERNEL_DISPATCH_SINGLE_PRODUCER` = 2, `AMD_DBGAPI_QUEUE_TYPE_AMD_PM4` = 3 }

*Queue type.*

- enum `amd_dbgapi_queue_state_t` { `AMD_DBGAPI_QUEUE_STATE_VALID` = 1, `AMD_DBGAPI_QUEUE_STATE_ERROR` = 2 }

*Queue state.*

- enum `amd_dbgapi_queue_error_reason_t` { `AMD_DBGAPI_QUEUE_ERROR_REASON_INVALID_PACKET` = (1 << 0), `AMD_DBGAPI_QUEUE_ERROR_REASON_MEMORY_VIOLATION` = (1 << 1), `AMD_DBGAPI_QUEUE_ERROR_REASON_ASSERT_TRAP` = (1 << 2), `AMD_DBGAPI_QUEUE_ERROR_REASON_WAVE_ERROR` = (1 << 3) }

*A bit mask of the reasons that a queue is in error.*

- enum `amd_dbgapi_dispatch_info_t` {  
`AMD_DBGAPI_DISPATCH_INFO_QUEUE` = 1, `AMD_DBGAPI_DISPATCH_INFO_AGENT` = 2, `AMD_DBGAPI_DISPATCH_INFO_ARCHITECTURE` = 3, `AMD_DBGAPI_DISPATCH_INFO_PACKET_ID` = 4,  
`AMD_DBGAPI_DISPATCH_INFO_BARRIER` = 5, `AMD_DBGAPI_DISPATCH_INFO_ACQUIRE_FENCE` = 6, `AMD_DBGAPI_DISPATCH_INFO_RELEASE_FENCE` = 7, `AMD_DBGAPI_DISPATCH_INFO_GRID_DIMENSIONS` = 8,  
`AMD_DBGAPI_DISPATCH_INFO_WORK_GROUP_SIZES` = 9, `AMD_DBGAPI_DISPATCH_INFO_GRID_SIZES` = 10, `AMD_DBGAPI_DISPATCH_INFO_PRIVATE_SEGMENT_SIZE` = 11, `AMD_DBGAPI_DISPATCH_INFO_GROUP_SEGMENT_SIZE` = 12,  
`AMD_DBGAPI_DISPATCH_INFO_KERNEL_ARGUMENT_SEGMENT_ADDRESS` = 13, `AMD_DBGAPI_DISPATCH_INFO_KERNEL_ENTRY_ADDRESS` = 14 }

*Dispatch queries that are supported by `amd_dbgapi_dispatch_get_info`.*

- enum `amd_dbgapi_dispatch_barrier_t` { `AMD_DBGAPI_DISPATCH_BARRIER_NONE` = 0, `AMD_DBGAPI_DISPATCH_BARRIER_PRESENT` = 1 }

*Dispatch barrier.*

- enum `amd_dbgapi_dispatch_fence_scope_t` { `AMD_DBGAPI_DISPATCH_FENCE_SCOPE_NONE` = 0, `AMD_DBGAPI_DISPATCH_FENCE_SCOPE_AGENT` = 1, `AMD_DBGAPI_DISPATCH_FENCE_SCOPE_SYSTEM` = 2 }

*Dispatch memory fence scope.*

- enum `amd_dbgapi_wave_info_t` {  
`AMD_DBGAPI_WAVE_INFO_STATE` = 1, `AMD_DBGAPI_WAVE_INFO_STOP_REASON` = 2, `AMD_DBGAPI_WAVE_INFO_WATCHPOINTS` = 3, `AMD_DBGAPI_WAVE_INFO_DISPATCH` = 4,  
`AMD_DBGAPI_WAVE_INFO_QUEUE` = 5, `AMD_DBGAPI_WAVE_INFO_AGENT` = 6, `AMD_DBGAPI_WAVE_INFO_ARCHITECTURE` = 7, `AMD_DBGAPI_WAVE_INFO_PC` = 8,  
`AMD_DBGAPI_WAVE_INFO_EXEC_MASK` = 9, `AMD_DBGAPI_WAVE_INFO_WORK_GROUP_COORD` = 10, `AMD_DBGAPI_WAVE_INFO_WAVE_NUMBER_IN_WORK_GROUP` = 11, `AMD_DBGAPI_WAVE_INFO_LANE_COUNT` = 12 }

*Wave queries that are supported by `amd_dbgapi_wave_get_info`.*

- enum `amd_dbgapi_wave_state_t` { `AMD_DBGAPI_WAVE_STATE_RUN` = 1, `AMD_DBGAPI_WAVE_STATE_SINGLE_STEP` = 2, `AMD_DBGAPI_WAVE_STATE_STOP` = 3 }

*The execution state of a wave.*

- enum `amd_dbgapi_wave_stop_reason_t` {  
`AMD_DBGAPI_WAVE_STOP_REASON_NONE` = 0, `AMD_DBGAPI_WAVE_STOP_REASON_BREAKPOINT` = (1 << 0), `AMD_DBGAPI_WAVE_STOP_REASON_WATCHPOINT` = (1 << 1), `AMD_DBGAPI_WAVE_STOP_REASON_SINGLE_STEP` = (1 << 2),  
`AMD_DBGAPI_WAVE_STOP_REASON_QUEUE_ERROR` = (1 << 3), `AMD_DBGAPI_WAVE_STOP_REASON_FP_INPUT_DENORMAL` = (1 << 4), `AMD_DBGAPI_WAVE_STOP_REASON_FP_DIVIDE_BY_0` = (1 << 5), `AMD_DBGAPI_WAVE_STOP_REASON_FP_OVERFLOW` = (1 << 6),  
`AMD_DBGAPI_WAVE_STOP_REASON_FP_UNDERFLOW` = (1 << 7), `AMD_DBGAPI_WAVE_STOP_REASON_FP_INEXACT` = (1 << 8), `AMD_DBGAPI_WAVE_STOP_REASON_FP_INVALID_OPERATION` = (1 << 9), `AMD_DBGAPI_WAVE_STOP_REASON_INT_DIVIDE_BY_0` = (1 << 10),  
`AMD_DBGAPI_WAVE_STOP_REASON_DEBUG_TRAP` = (1 << 11), `AMD_DBGAPI_WAVE_STOP_REASON_ASSERT_TRAP` = (1 << 12), `AMD_DBGAPI_WAVE_STOP_REASON_TRAP` = (1 << 13), `AMD_DBGAPI_WAVE_STOP_REASON_MEMORY_VIOLATION` = (1 << 14),  
`AMD_DBGAPI_WAVE_STOP_REASON_ILLEGAL_INSTRUCTION` = (1 << 15), `AMD_DBGAPI_WAVE_STOP_REASON_ECC_ERROR` = (1 << 16), `AMD_DBGAPI_WAVE_STOP_REASON_FATAL_HALT` = (1 << 17), `AMD_DBGAPI_WAVE_STOP_REASON_XNACK_ERROR` = (1 << 18) }

*A bit mask of the reasons that a wave stopped.*

- enum `amd_dbgapi_resume_mode_t` { `AMD_DBGAPI_RESUME_MODE_NORMAL` = 0, `AMD_DBGAPI_RESUME_MODE_SINGLE_STEP` = 1 }

*The mode in which to resuming the execution of a wave.*

- enum `amd_dbgapi_watchpoint_share_kind_t` { `AMD_DBGAPI_WATCHPOINT_SHARE_KIND_UNSUPPORTED` = 0, `AMD_DBGAPI_WATCHPOINT_SHARE_KIND_UNSHARED` = 1, `AMD_DBGAPI_WATCHPOINT_SHARE_KIND_SHARED` = 2 }

*The way watchpoints are shared between processes.*

- enum `amd_dbgapi_watchpoint_kind_t` { `AMD_DBGAPI_WATCHPOINT_KIND_LOAD` = 1, `AMD_DBGAPI_WATCHPOINT_KIND_STORE_AND_RMW` = 2, `AMD_DBGAPI_WATCHPOINT_KIND_RMW` = 3, `AMD_DBGAPI_WATCHPOINT_KIND_ALL` = 4 }

*Watchpoint memory access kinds.*

- enum `amd_dbgapi_register_class_info_t` { `AMD_DBGAPI_REGISTER_CLASS_INFO_NAME` = 1 }

*Register class queries that are supported by `amd_dbgapi_architecture_register_class_get_info`.*

- enum `amd_dbgapi_register_info_t` { `AMD_DBGAPI_REGISTER_INFO_NAME` = 1, `AMD_DBGAPI_REGISTER_INFO_SIZE` = 2, `AMD_DBGAPI_REGISTER_INFO_TYPE` = 3 }

*Register queries that are supported by `amd_dbgapi_architecture_register_get_info` and `amd_dbgapi_wave_register_get_info`.*

- enum `amd_dbgapi_register_class_state_t` { `AMD_DBGAPI_REGISTER_CLASS_STATE_NOT_MEMBER` = 0, `AMD_DBGAPI_REGISTER_CLASS_STATE_MEMBER` = 1 }

*Indication of whether a register is a member of a register class.*

- enum `amd_dbgapi_address_class_info_t` { `AMD_DBGAPI_ADDRESS_CLASS_INFO_NAME` = 1, `AMD_DBGAPI_ADDRESS_CLASS_INFO_ADDRESS_SPACE` = 2 }

Source language address class queries that are supported by [amd\\_dbgapi\\_architecture\\_address\\_class\\_get\\_info](#).

- enum [amd\\_dbgapi\\_address\\_space\\_access\\_t](#) { [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_ACCESS\\_ALL](#) = 1, [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_ACCESS\\_PROGRAM\\_CONSTANT](#) = 2, [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_ACCESS\\_DISPATCH\\_CONSTANT](#) = 3 }

Indication of how the address space is accessed.

- enum [amd\\_dbgapi\\_address\\_space\\_info\\_t](#) { [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_INFO\\_NAME](#) = 1, [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_INFO\\_ADDRESS\\_SIZE](#) = 2, [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_INFO\\_NULL\\_ADDRESS](#) = 3, [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_INFO\\_ACCESS](#) = 4 }

Address space queries that are supported by [amd\\_dbgapi\\_address\\_space\\_get\\_info](#).

- enum [amd\\_dbgapi\\_address\\_space\\_alias\\_t](#) { [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_ALIAS\\_NONE](#) = 0, [AMD\\_DBGAPI\\_ADDRESS\\_SPACE\\_ALIAS\\_MAY](#) = 1 }

Indication of whether addresses in two address spaces may alias.

- enum [amd\\_dbgapi\\_address\\_class\\_state\\_t](#) { [AMD\\_DBGAPI\\_ADDRESS\\_CLASS\\_STATE\\_NOT\\_MEMBER](#) = 0, [AMD\\_DBGAPI\\_ADDRESS\\_CLASS\\_STATE\\_MEMBER](#) = 1 }

Indication of whether a segment address in an address space is a member of an source language address class.

- enum [amd\\_dbgapi\\_memory\\_precision\\_t](#) { [AMD\\_DBGAPI\\_MEMORY\\_PRECISION\\_NONE](#) = 0, [AMD\\_DBGAPI\\_MEMORY\\_PRECISION\\_PRECISE](#) = 1 }

Memory access precision.

- enum [amd\\_dbgapi\\_event\\_kind\\_t](#) { [AMD\\_DBGAPI\\_EVENT\\_KIND\\_NONE](#) = 0, [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_STOP](#) = 1, [AMD\\_DBGAPI\\_EVENT\\_KIND\\_WAVE\\_COMMAND\\_TERMINATED](#) = 2, [AMD\\_DBGAPI\\_EVENT\\_KIND\\_CODE\\_OBJECT\\_LIST\\_UPDATED](#) = 3, [AMD\\_DBGAPI\\_EVENT\\_KIND\\_BREAKPOINT\\_RESUME](#) = 4, [AMD\\_DBGAPI\\_EVENT\\_KIND\\_RUNTIME](#) = 5, [AMD\\_DBGAPI\\_EVENT\\_KIND\\_QUEUE\\_ERROR](#) = 6 }

The event kinds.

- enum [amd\\_dbgapi\\_runtime\\_state\\_t](#) { [AMD\\_DBGAPI\\_RUNTIME\\_STATE\\_LOADED\\_SUPPORTED](#) = 1, [AMD\\_DBGAPI\\_RUNTIME\\_STATE\\_LOADED\\_UNSUPPORTED](#) = 2, [AMD\\_DBGAPI\\_RUNTIME\\_STATE\\_UNLOADED](#) = 3 }

Inferior runtime state.

- enum [amd\\_dbgapi\\_event\\_info\\_t](#) { [AMD\\_DBGAPI\\_EVENT\\_INFO\\_KIND](#) = 1, [AMD\\_DBGAPI\\_EVENT\\_INFO\\_WAVE](#) = 2, [AMD\\_DBGAPI\\_EVENT\\_INFO\\_BREAKPOINT](#) = 3, [AMD\\_DBGAPI\\_EVENT\\_INFO\\_CLIENT\\_THREAD](#) = 4, [AMD\\_DBGAPI\\_EVENT\\_INFO\\_RUNTIME\\_STATE](#) = 5, [AMD\\_DBGAPI\\_EVENT\\_INFO\\_RUNTIME\\_VERSION](#) = 6 }

Event queries that are supported by [amd\\_dbgapi\\_event\\_get\\_info](#).

- enum [amd\\_dbgapi\\_log\\_level\\_t](#) { [AMD\\_DBGAPI\\_LOG\\_LEVEL\\_NONE](#) = 0, [AMD\\_DBGAPI\\_LOG\\_LEVEL\\_FATAL\\_ERROR](#) = 1, [AMD\\_DBGAPI\\_LOG\\_LEVEL\\_WARNING](#) = 2, [AMD\\_DBGAPI\\_LOG\\_LEVEL\\_INFO](#) = 3, [AMD\\_DBGAPI\\_LOG\\_LEVEL\\_VERBOSE](#) = 4 }

The logging levels supported.

- enum [amd\\_dbgapi\\_shared\\_library\\_state\\_t](#) { [AMD\\_DBGAPI\\_SHARED\\_LIBRARY\\_STATE\\_LOADED](#) = 1, [AMD\\_DBGAPI\\_SHARED\\_LIBRARY\\_STATE\\_UNLOADED](#) = 2 }

The state of a shared library.

- enum [amd\\_dbgapi\\_breakpoint\\_action\\_t](#) { [AMD\\_DBGAPI\\_BREAKPOINT\\_ACTION\\_RESUME](#) = 1, [AMD\\_DBGAPI\\_BREAKPOINT\\_ACTION\\_HALT](#) = 2 }

The action to perform after reporting a breakpoint has been hit.

- enum [amd\\_dbgapi\\_breakpoint\\_state\\_t](#) { [AMD\\_DBGAPI\\_BREAKPOINT\\_STATE\\_DISABLE](#) = 1, [AMD\\_DBGAPI\\_BREAKPOINT\\_STATE\\_ENABLE](#) = 2 }

The state of a breakpoint.

## Functions

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_get_status_string (amd_dbgapi_status_t status, const char **status_string)`  
*Query a textual description of a status code.*
- `void AMD_DBGAPI amd_dbgapi_get_version (uint32_t *major, uint32_t *minor, uint32_t *patch)`  
*Query the version of the installed library.*
- `const char AMD_DBGAPI * amd_dbgapi_get_build_name (void)`  
*Query the installed library build name.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_initialize (amd_dbgapi_callbacks_t *callbacks)`  
*Initialize the library.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_finalize (void)`  
*Finalize the library.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_architecture_get_info (amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_architecture_info_t query, size_t value_size, void *value)`  
*Query information about an architecture.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_get_architecture (uint32_t elf_amdgpu_machine, amd_dbgapi_architecture_id_t *architecture_id)`  
*Get an architecture from the AMD GPU ELF `EF_AMDGPU_MACH` value corresponding to the architecture.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_disassemble_instruction (amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_global_address_t address, amd_dbgapi_size_t *size, const void *memory, char **instruction_text, size_t *address_operand_count, amd_dbgapi_global_address_t **address_operands)`  
*Disassemble a single instruction.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_get_info (amd_dbgapi_process_id_t process_id, amd_dbgapi_process_info_t query, size_t value_size, void *value)`  
*Query information about a process.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_attach (amd_dbgapi_client_process_id_t client_process_id, amd_dbgapi_process_id_t *process_id)`  
*Attach to a process in order to provide debug control of the AMD GPUs it uses.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_detach (amd_dbgapi_process_id_t process_id)`  
*Detach from a process.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_set_progress (amd_dbgapi_process_id_t process_id, amd_dbgapi_progress_t progress)`  
*Set the progress required for a process.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_code_object_get_info (amd_dbgapi_process_id_t process_id, amd_dbgapi_code_object_id_t code_object_id, amd_dbgapi_code_object_info_t query, size_t value_size, void *value)`  
*Query information about a code object.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_code_object_list (amd_dbgapi_process_id_t process_id, size_t *code_object_count, amd_dbgapi_code_object_id_t **code_objects, amd_dbgapi_changed_t *changed)`  
*Return the list of loaded code objects for a process.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_agent_get_info (amd_dbgapi_process_id_t process_id, amd_dbgapi_agent_id_t agent_id, amd_dbgapi_agent_info_t query, size_t value_size, void *value)`  
*Query information about an agent.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_agent_list (amd_dbgapi_process_id_t process_id, size_t *agent_count, amd_dbgapi_agent_id_t **agents, amd_dbgapi_changed_t *changed)`  
*Return the list of agents for a process.*
- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_queue_get_info (amd_dbgapi_process_id_t process_id, amd_dbgapi_queue_id_t queue_id, amd_dbgapi_queue_info_t query, size_t value_size, void *value)`



*Query information about a queue.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_queue_list (amd_dbgapi_process_id_t process_id, size_t *queue_count, amd_dbgapi_queue_id_t **queues, amd_dbgapi_changed_t *changed)`

*Return the list of queues for a process.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_queue_packet_list (amd_dbgapi_process_id_t process_id, amd_dbgapi_queue_id_t queue_id, amd_dbgapi_queue_packet_id_t *first_packet_id, amd_dbgapi_size_t *packets_byte_size, void **packets_bytes)`

*Return the packets for a queue of a process.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_dispatch_get_info (amd_dbgapi_process_id_t process_id, amd_dbgapi_dispatch_id_t dispatch_id, amd_dbgapi_dispatch_info_t query, size_t value_size, void *value)`

*Query information about a dispatch.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_dispatch_list (amd_dbgapi_process_id_t process_id, size_t *dispatch_count, amd_dbgapi_dispatch_id_t **dispatches, amd_dbgapi_changed_t *changed)`

*Return the list of dispatches for a process.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_wave_get_info (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_wave_info_t query, size_t value_size, void *value)`

*Query information about a wave.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_wave_list (amd_dbgapi_process_id_t process_id, size_t *wave_count, amd_dbgapi_wave_id_t **waves, amd_dbgapi_changed_t *changed)`

*Return the list of existing waves for a process.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_wave_stop (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id)`

*Request a wave to stop executing.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_wave_resume (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_resume_mode_t resume_mode)`

*Resume execution of a stopped wave.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_displaced_stepping_start (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, const void *saved_instruction_bytes, amd_dbgapi_displaced_stepping_id_t *displaced_stepping)`

*Create a displaced stepping buffer.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_displaced_stepping_complete (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_id_t wave_id, amd_dbgapi_displaced_stepping_id_t displaced_stepping)`

*Complete a displaced stepping buffer for a wave.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_set_watchpoint (amd_dbgapi_process_id_t process_id, amd_dbgapi_agent_id_t agent_id, amd_dbgapi_global_address_t address, amd_dbgapi_size_t size, amd_dbgapi_watchpoint_kind_t kind, amd_dbgapi_watchpoint_id_t *watchpoint_id, amd_dbgapi_global_address_t *watchpoint_address, amd_dbgapi_size_t *watchpoint_size)`

*Set a hardware data watchpoint.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_remove_watchpoint (amd_dbgapi_process_id_t process_id, amd_dbgapi_agent_id_t agent_id, amd_dbgapi_watchpoint_id_t watchpoint_id)`

*Remove a hardware data watchpoint previously set by `amd_dbgapi_set_watchpoint`.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_architecture_register_class_get_info (amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_register_class_id_t register_class_id, amd_dbgapi_register_class_info_t query, size_t value_size, void *value)`

*Query information about a register class of an architecture.*

- `amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_architecture_register_class_list (amd_dbgapi_architecture_id_t architecture_id, size_t *register_class_count, amd_dbgapi_register_class_id_t **register_classes)`

*Report the list of register classes supported by the architecture.*

- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_architecture\\_register\\_get\\_info](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_register\\_info\\_t](#) query, [size\\_t](#) value\_size, void \*value)  
*Query information about a register of an architecture.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_wave\\_register\\_get\\_info](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_register\\_info\\_t](#) query, [size\\_t](#) value\_size, void \*value)  
*Query information about a register of a wave.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_architecture\\_register\\_list](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [size\\_t](#) \*register\_count, [amd\\_dbgapi\\_register\\_id\\_t](#) \*\*registers)  
*Report the list of registers supported by the architecture.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_wave\\_register\\_list](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [size\\_t](#) \*register\_count, [amd\\_dbgapi\\_register\\_id\\_t](#) \*\*registers)  
*Report the list of registers supported by a wave.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_dwarf\\_register\\_to\\_register](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [uint64\\_t](#) dwarf\_register, [amd\\_dbgapi\\_register\\_id\\_t](#) \*register\_id)  
*Return a register handle from an AMD GPU DWARF register number.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_register\\_is\\_in\\_register\\_class](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_register\\_class\\_id\\_t](#) register\_class\_id, [amd\\_dbgapi\\_register\\_class\\_state\\_t](#) \*register\_class\_state)  
*Determine if a register is a member of a register class.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_read\\_register](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_size\\_t](#) offset, [amd\\_dbgapi\\_size\\_t](#) value\_size, void \*value)  
*Read a register.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_write\\_register](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_size\\_t](#) offset, [amd\\_dbgapi\\_size\\_t](#) value\_size, const void \*value)  
*Write a register.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_prefetch\\_register](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_register\\_id\\_t](#) register\_id, [amd\\_dbgapi\\_size\\_t](#) register\_count)  
*Prefetch register values.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_architecture\\_address\\_class\\_get\\_info](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_address\\_class\\_id\\_t](#) address\_class\_id, [amd\\_dbgapi\\_address\\_class\\_info\\_t](#) query, [size\\_t](#) value\_size, void \*value)  
*Query information about a source language address class of an architecture.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_architecture\\_address\\_class\\_list](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [size\\_t](#) \*address\_class\_count, [amd\\_dbgapi\\_address\\_class\\_id\\_t](#) \*\*address\_classes)  
*Report the list of source language address classes supported by the architecture.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_dwarf\\_address\\_class\\_to\\_address\\_class](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [uint64\\_t](#) dwarf\_address\_class, [amd\\_dbgapi\\_address\\_class\\_id\\_t](#) \*address\_class\_id)  
*Return the architecture source language address class from a DWARF address class number.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_address\\_space\\_get\\_info](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id, [amd\\_dbgapi\\_address\\_space\\_info\\_t](#) query, [size\\_t](#) value\_size, void \*value)  
*Query information about an address space.*
- [amd\\_dbgapi\\_status\\_t AMD\\_DBGAPI amd\\_dbgapi\\_architecture\\_address\\_space\\_list](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [size\\_t](#) \*address\_space\_count, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) \*\*address\_spaces)  
*Report the list of address spaces supported by the architecture.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_dwarf\\_address\\_space\\_to\\_address\\_space](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [uint64\\_t](#) dwarf\_address\_space, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) \*address\_space\_id)

*Return the address space from an AMD GPU DWARF address space number.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_address\\_spaces\\_may\\_alias](#) ([amd\\_dbgapi\\_architecture\\_id\\_t](#) architecture\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id1, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id2, [amd\\_dbgapi\\_address\\_space\\_alias\\_t](#) \*address\_space\_alias)

*Determine if an address in one address space may alias an address in another address space.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_convert\\_address\\_space](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_lane\\_id\\_t](#) lane\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) source\_address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) source\_segment\_address, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) destination\_address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) \*destination\_segment\_address)

*Convert a source segment address in the source address space into a destination segment address in the destination address space.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_address\\_is\\_in\\_address\\_class](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_lane\\_id\\_t](#) lane\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) segment\_address, [amd\\_dbgapi\\_address\\_class\\_id\\_t](#) address\_class\_id, [amd\\_dbgapi\\_address\\_class\\_state\\_t](#) \*address\_class\_state)

*Determine if a segment address in an address space is a member of a source language address class.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_read\\_memory](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_lane\\_id\\_t](#) lane\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) segment\_address, [amd\\_dbgapi\\_size\\_t](#) \*value\_size, void \*value)

*Read memory.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_write\\_memory](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_wave\\_id\\_t](#) wave\_id, [amd\\_dbgapi\\_lane\\_id\\_t](#) lane\_id, [amd\\_dbgapi\\_address\\_space\\_id\\_t](#) address\_space\_id, [amd\\_dbgapi\\_segment\\_address\\_t](#) segment\_address, [amd\\_dbgapi\\_size\\_t](#) \*value\_size, const void \*value)

*Write memory.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_set\\_memory\\_precision](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_agent\\_id\\_t](#) agent\_id, [amd\\_dbgapi\\_memory\\_precision\\_t](#) memory\_precision)

*Control precision of memory access reporting.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_next\\_pending\\_event](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_event\\_id\\_t](#) \*event\_id, [amd\\_dbgapi\\_event\\_kind\\_t](#) \*kind)

*Obtain the next pending event for a process.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_event\\_get\\_info](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_event\\_id\\_t](#) event\_id, [amd\\_dbgapi\\_event\\_info\\_t](#) query, [size\\_t](#) value\_size, void \*value)

*Query information about an event.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_event\\_processed](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_event\\_id\\_t](#) event\_id)

*Report that an event has been processed.*

- void [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_set\\_log\\_level](#) ([amd\\_dbgapi\\_log\\_level\\_t](#) level)

*Set the logging level.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_report\\_shared\\_library](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_shared\\_library\\_id\\_t](#) shared\_library\_id, [amd\\_dbgapi\\_shared\\_library\\_state\\_t](#) shared\_library\_state)

*Report that a shared library enabled by the [amd\\_dbgapi\\_callbacks\\_s::enable\\_notify\\_shared\\_library](#) callback has been loaded or unloaded.*

- [amd\\_dbgapi\\_status\\_t](#) [AMD\\_DBGAPI](#) [amd\\_dbgapi\\_report\\_breakpoint\\_hit](#) ([amd\\_dbgapi\\_process\\_id\\_t](#) process\_id, [amd\\_dbgapi\\_breakpoint\\_id\\_t](#) breakpoint\_id, [amd\\_dbgapi\\_client\\_thread\\_id\\_t](#) client\_thread\_id, [amd\\_dbgapi\\_breakpoint\\_action\\_t](#) \*breakpoint\_action)

*Report that a breakpoint added by the [amd\\_dbgapi\\_callbacks\\_s::add\\_breakpoint](#) callback has been hit.*

#### 4.1.1 Detailed Description

AMD debugger API interface.

#### 4.1.2 Macro Definition Documentation

4.1.2.1 `#define AMD_DBGAPI_AMD_DBGAPI_IMPORT`

4.1.2.2 `#define AMD_DBGAPI_CALL`

4.1.2.3 `#define AMD_DBGAPI_EXPORT_AMD_DBGAPI_EXPORT_DECORATOR_AMD_DBGAPI_CALL`

4.1.2.4 `#define AMD_DBGAPI_IMPORT_AMD_DBGAPI_IMPORT_DECORATOR_AMD_DBGAPI_CALL`

# Index

- AMD\_DBGAPI\_ADDRESS\_CLASS\_INFO\_ADDRESS\_SPACE
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_CLASS\_INFO\_NAME
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_CLASS\_STATE\_MEMBER
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_CLASS\_STATE\_NOT\_MEMBER
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_ALL
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_DISPATCH\_CONSTANT
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_PROGRAM\_CONSTANT
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_ALIAS\_MAY
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_ALIAS\_NONE
  - Memory, [76](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_ACCESS
  - Memory, [77](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_ADDRESS\_SIZE
  - Memory, [77](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_NAME
  - Memory, [77](#)
- AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_NULL\_ADDRESS
  - Memory, [77](#)
- AMD\_DBGAPI\_AGENT\_INFO\_ARCHITECTURE
  - Agents, [29](#)
- AMD\_DBGAPI\_AGENT\_INFO\_COMPUTE\_UNIT\_COUNT
  - Agents, [29](#)
- AMD\_DBGAPI\_AGENT\_INFO\_MAX\_WAVES\_PER\_SIMD
  - Agents, [29](#)
- AMD\_DBGAPI\_AGENT\_INFO\_NAME
  - Agents, [29](#)
- AMD\_DBGAPI\_AGENT\_INFO\_NUM\_SIMD\_PER\_COMPUTE\_UNIT
  - Agents, [29](#)
- AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_DEVICE\_ID
  - Agents, [29](#)
- AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_SLOT
  - Agents, [29](#)
- AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_VENDOR\_ID
  - Agents, [29](#)
- AMD\_DBGAPI\_AGENT\_INFO\_SHADER\_ENGINE\_COUNT
  - Agents, [29](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION\_PC\_ADJUST
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION\_SIZE
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_DEFAULT\_GLOBAL\_ADDRESS\_SPACE
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_ELF\_AMDGPU\_MACHINE
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_EXECUTION\_MASK\_REGISTER
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_LARGEST\_INSTRUCTION\_SIZE
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_MINIMUM\_INSTRUCTION\_ALIGNMENT
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_NAME
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_PC\_REGISTER
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_PRECISE\_MEMORY\_SUPPORTED
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_WATCHPOINT\_COUNT
  - Architectures, [15](#)
- AMD\_DBGAPI\_ARCHITECTURE\_INFO\_WATCHPOINT\_SHARE
  - Architectures, [15](#)

- Architectures, [15](#)
- AMD\_DBGAPI\_BREAKPOINT\_ACTION\_HALT
  - Callbacks, [99](#)
- AMD\_DBGAPI\_BREAKPOINT\_ACTION\_RESUME
  - Callbacks, [99](#)
- AMD\_DBGAPI\_BREAKPOINT\_STATE\_DISABLE
  - Callbacks, [100](#)
- AMD\_DBGAPI\_BREAKPOINT\_STATE\_ENABLE
  - Callbacks, [100](#)
- AMD\_DBGAPI\_CHANGED\_NO
  - Basic Types, [6](#)
- AMD\_DBGAPI\_CHANGED\_YES
  - Basic Types, [6](#)
- AMD\_DBGAPI\_CODE\_OBJECT\_INFO\_LOAD\_ADDRESS
  - Code Objects, [25](#)
- AMD\_DBGAPI\_CODE\_OBJECT\_INFO\_URI\_NAME
  - Code Objects, [25](#)
- AMD\_DBGAPI\_DISPATCH\_BARRIER\_NONE
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_BARRIER\_PRESENT
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_FENCE\_SCOPE\_AGENT
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_FENCE\_SCOPE\_NONE
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_FENCE\_SCOPE\_SYSTEM
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_ACQUIRE\_FENCE
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_AGENT
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_ARCHITECTURE
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_BARRIER
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_GRID\_DIMENSIONS
  - Dispatches, [41](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_GRID\_SIZES
  - Dispatches, [41](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_GROUP\_SEGMENT\_SIZE
  - Dispatches, [41](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_KERNEL\_ARGUMENT\_SEGMENT\_ADDRESS
  - Dispatches, [41](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_KERNEL\_ENTRY\_ADDRESS
  - Dispatches, [41](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_PACKET\_ID
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_PRIVATE\_SEGMENT\_SIZE
  - Dispatches, [41](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_QUEUE
  - Dispatches, [40](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_RELEASE\_FENCE
  - Dispatches, [41](#)
- AMD\_DBGAPI\_DISPATCH\_INFO\_WORK\_GROUP\_SIZES
  - Dispatches, [41](#)
- AMD\_DBGAPI\_EVENT\_INFO\_BREAKPOINT
  - Events, [90](#)
- AMD\_DBGAPI\_EVENT\_INFO\_CLIENT\_THREAD
  - Events, [90](#)
- AMD\_DBGAPI\_EVENT\_INFO\_KIND
  - Events, [90](#)
- AMD\_DBGAPI\_EVENT\_INFO\_RUNTIME\_STATE
  - Events, [90](#)
- AMD\_DBGAPI\_EVENT\_INFO\_RUNTIME\_VERSION
  - Events, [90](#)
- AMD\_DBGAPI\_EVENT\_INFO\_WAVE
  - Events, [90](#)
- AMD\_DBGAPI\_EVENT\_KIND\_BREAKPOINT\_RESUME
  - Events, [91](#)
- AMD\_DBGAPI\_EVENT\_KIND\_CODE\_OBJECT\_LIST\_UPDATED
  - Events, [91](#)
- AMD\_DBGAPI\_EVENT\_KIND\_NONE
  - Events, [91](#)
- AMD\_DBGAPI\_EVENT\_KIND\_QUEUE\_ERROR
  - Events, [91](#)
- AMD\_DBGAPI\_EVENT\_KIND\_RUNTIME
  - Events, [91](#)
- AMD\_DBGAPI\_EVENT\_KIND\_WAVE\_COMMAND\_TERMINATED
  - Events, [91](#)
- AMD\_DBGAPI\_EVENT\_KIND\_WAVE\_STOP
  - Events, [91](#)
- AMD\_DBGAPI\_LOG\_LEVEL\_FATAL\_ERROR
  - Logging, [95](#)
- AMD\_DBGAPI\_LOG\_LEVEL\_INFO
  - Logging, [95](#)
- AMD\_DBGAPI\_LOG\_LEVEL\_NONE
  - Logging, [95](#)
- AMD\_DBGAPI\_LOG\_LEVEL\_VERBOSE
  - Logging, [95](#)
- AMD\_DBGAPI\_LOG\_LEVEL\_WARNING
  - Logging, [95](#)
- AMD\_DBGAPI\_MEMORY\_PRECISION\_NONE
  - Memory, [77](#)
- AMD\_DBGAPI\_MEMORY\_PRECISION\_PRECISE
  - Memory, [77](#)
- AMD\_DBGAPI\_PROCESS\_INFO\_NOTIFIER
  - Processes, [20](#)
- AMD\_DBGAPI\_PROGRESS\_NO\_FORWARD
  - Processes, [20](#)
- AMD\_DBGAPI\_PROGRESS\_NORMAL

- Processes, [20](#)
- AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_ASSERT\_TRAP
  - Queues, [33](#)
- AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_INVALID\_PACKET
  - Queues, [33](#)
- AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_MEMORY\_VIOLATION
  - Queues, [33](#)
- AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_WAVE\_ERROR
  - Queues, [33](#)
- AMD\_DBGAPI\_QUEUE\_INFO\_AGENT
  - Queues, [33](#)
- AMD\_DBGAPI\_QUEUE\_INFO\_ARCHITECTURE
  - Queues, [33](#)
- AMD\_DBGAPI\_QUEUE\_INFO\_ERROR\_REASON
  - Queues, [34](#)
- AMD\_DBGAPI\_QUEUE\_INFO\_STATE
  - Queues, [34](#)
- AMD\_DBGAPI\_QUEUE\_STATE\_ERROR
  - Queues, [34](#)
- AMD\_DBGAPI\_QUEUE\_STATE\_VALID
  - Queues, [34](#)
- AMD\_DBGAPI\_QUEUE\_TYPE
  - Queues, [34](#)
- AMD\_DBGAPI\_QUEUE\_TYPE\_AMD\_PM4
  - Queues, [34](#)
- AMD\_DBGAPI\_QUEUE\_TYPE\_HSA\_KERNEL\_DISPATCH\_MULTIPLE\_PRODUCER
  - Queues, [34](#)
- AMD\_DBGAPI\_QUEUE\_TYPE\_HSA\_KERNEL\_DISPATCH\_SINGLE\_PRODUCER
  - Queues, [34](#)
- AMD\_DBGAPI\_QUEUE\_TYPE\_UNKNOWN
  - Queues, [34](#)
- AMD\_DBGAPI\_REGISTER\_CLASS\_INFO\_NAME
  - Registers, [63](#)
- AMD\_DBGAPI\_REGISTER\_CLASS\_STATE\_MEMBER
  - Registers, [63](#)
- AMD\_DBGAPI\_REGISTER\_CLASS\_STATE\_NOT\_MEMBER
  - Registers, [63](#)
- AMD\_DBGAPI\_REGISTER\_INFO\_NAME
  - Registers, [63](#)
- AMD\_DBGAPI\_REGISTER\_INFO\_SIZE
  - Registers, [63](#)
- AMD\_DBGAPI\_REGISTER\_INFO\_TYPE
  - Registers, [63](#)
- AMD\_DBGAPI\_RESUME\_MODE\_NORMAL
  - Wave, [44](#)
- AMD\_DBGAPI\_RESUME\_MODE\_SINGLE\_STEP
  - Wave, [44](#)
- AMD\_DBGAPI\_RUNTIME\_STATE\_LOADED\_SUPPORTED
  - Events, [92](#)
- AMD\_DBGAPI\_RUNTIME\_STATE\_LOADED\_UNSUPPORTED
  - Events, [92](#)
- AMD\_DBGAPI\_RUNTIME\_STATE\_UNLOADED
  - Events, [92](#)
- AMD\_DBGAPI\_SHARED\_LIBRARY\_STATE\_LOADED
  - Callbacks, [100](#)
- AMD\_DBGAPI\_SHARED\_LIBRARY\_STATE\_UNLOADED
  - Callbacks, [100](#)
- AMD\_DBGAPI\_STATUS\_ERROR
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_ALREADY\_ATTACHED
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_ALREADY\_INITIALIZED
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_CLIENT\_CALLBACK
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_DISPLACED\_STEPPING\_BUFFER\_UNAVAILABLE
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_ILLEGAL\_INSTRUCTION
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_CLASS\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_SPACE\_CONVERSION
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_SPACE\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_AGENT\_ID
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARCHITECTURE\_ID
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARGUMENT
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARGUMENT\_SIZE
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_BREAKPOINT\_ID
  - Status Codes, [9](#)



- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_CLIENT\_PROCESS\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_CODE\_OBJECT\_ID
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_DISPATCH\_ID
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_DISPLACED\_STEPPING\_ID
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ELF\_AMD\_GPU\_MACHINE
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_EVENT\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_LANE\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_PROCESS\_ID
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_QUEUE\_ID
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_REGISTER\_CLASS\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_REGISTER\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_SHARED\_LIBRARY\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_WATCHPOINT\_ID
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_WAVE\_ID
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_LIBRARY\_NOT\_LOADED
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_MEMORY\_ACCESS
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_NO\_WATCHPOINT\_AVAILABLE
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_NOT\_INITIALIZED
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_NOT\_SUPPORTED
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_PROCESS\_EXITED
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_SYMBOL\_NOT\_FOUND
  - Status Codes, [9](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_VERSION\_MISMATCH
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_NOT\_RESUMABLE
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_NOT\_STOPPED
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_OUTSTANDING\_STOP
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_STOPPED
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_FATAL
  - Status Codes, [8](#)
- AMD\_DBGAPI\_STATUS\_SUCCESS
  - Status Codes, [8](#)
- AMD\_DBGAPI\_VERSION\_MAJOR
  - Versioning, [10](#)
- AMD\_DBGAPI\_VERSION\_MINOR
  - Versioning, [10](#)
- AMD\_DBGAPI\_WATCHPOINT\_KIND\_ALL
  - Watchpoints, [57](#)
- AMD\_DBGAPI\_WATCHPOINT\_KIND\_LOAD
  - Watchpoints, [57](#)
- AMD\_DBGAPI\_WATCHPOINT\_KIND\_RMW
  - Watchpoints, [57](#)
- AMD\_DBGAPI\_WATCHPOINT\_KIND\_STORE\_AND\_RMW
  - Watchpoints, [57](#)
- AMD\_DBGAPI\_WATCHPOINT\_SHARE\_KIND\_SHARED
  - Watchpoints, [57](#)
- AMD\_DBGAPI\_WATCHPOINT\_SHARE\_KIND\_UNSHARED
  - Watchpoints, [57](#)
- AMD\_DBGAPI\_WATCHPOINT\_SHARE\_KIND\_UNSUPPORTED
  - Watchpoints, [57](#)
- AMD\_DBGAPI\_WAVE\_INFO\_AGENT
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_INFO\_ARCHITECTURE
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_INFO\_DISPATCH
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_INFO\_EXEC\_MASK
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_INFO\_LANE\_COUNT
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_INFO\_PC
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_INFO\_QUEUE
  - Wave, [45](#)



- AMD\_DBGAPI\_WAVE\_INFO\_STATE
  - Wave, [44](#)
- AMD\_DBGAPI\_WAVE\_INFO\_STOP\_REASON
  - Wave, [44](#)
- AMD\_DBGAPI\_WAVE\_INFO\_WATCHPOINTS
  - Wave, [44](#)
- AMD\_DBGAPI\_WAVE\_INFO\_WAVE\_NUMBER\_IN\_WORK\_GROUP
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_INFO\_WORK\_GROUP\_COORD
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_STATE\_RUN
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_STATE\_SINGLE\_STEP
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_STATE\_STOP
  - Wave, [45](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ASSERT\_TRAP
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_BREAKPOINT
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_DEBUG\_TRAP
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ECC\_ERROR
  - Wave, [47](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FATAL\_HALT
  - Wave, [47](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_DIVIDE\_BY\_0
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INEXACT
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INPUT\_DENORMAL
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INVALID\_OPERATION
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_OVERFLOW
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_UNDERFLOW
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ILLEGAL\_INSTRUCTION
  - Wave, [47](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_INT\_DIVIDE\_BY\_0
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_MEMORY\_VIOLATION
  - Wave, [47](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_NONE
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_QUEUE\_ERROR
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_SINGLE\_STEP
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_TRAP
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_WATCHPOINT
  - Wave, [46](#)
- AMD\_DBGAPI\_WAVE\_STOP\_REASON\_XNACK\_ERROR
  - Wave, [47](#)
- AMD\_DBGAPI
  - amd-dbgapi.h, [128](#)
- AMD\_DBGAPI\_CALL
  - amd-dbgapi.h, [128](#)
- AMD\_DBGAPI\_EXPORT
  - amd-dbgapi.h, [128](#)
- AMD\_DBGAPI\_IMPORT
  - amd-dbgapi.h, [128](#)
- add\_breakpoint
  - amd\_dbgapi\_callbacks\_s, [107](#)
- Agents, [28](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_ARCHITECTURE, [29](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_COMPUTE\_UNIT\_COUNT, [29](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_MAX\_WAVES\_PER\_SIMD, [29](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_NAME, [29](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_NUM\_SIMD\_PER\_COMPUTE\_UNIT, [29](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_DEVICE\_ID, [29](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_SLOT, [29](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_PCIE\_VENDOR\_ID, [29](#)
  - AMD\_DBGAPI\_AGENT\_INFO\_SHADER\_ENGINE\_COUNT, [29](#)
  - amd\_dbgapi\_agent\_get\_info, [29](#)
  - amd\_dbgapi\_agent\_info\_t, [29](#)
  - amd\_dbgapi\_agent\_list, [30](#)
- allocate\_memory
  - amd\_dbgapi\_callbacks\_s, [107](#)
- amd-dbgapi.h
  - AMD\_DBGAPI, [128](#)
  - AMD\_DBGAPI\_CALL, [128](#)
  - AMD\_DBGAPI\_EXPORT, [128](#)
  - AMD\_DBGAPI\_IMPORT, [128](#)
- amd\_dbgapi\_address\_class\_id\_t, [103](#)
  - handle, [103](#)
- amd\_dbgapi\_address\_class\_info\_t

- Memory, [75](#)
- amd\_dbgapi\_address\_class\_state\_t
  - Memory, [76](#)
- amd\_dbgapi\_address\_is\_in\_address\_class
  - Memory, [77](#)
- amd\_dbgapi\_address\_space\_access\_t
  - Memory, [76](#)
- amd\_dbgapi\_address\_space\_alias\_t
  - Memory, [76](#)
- amd\_dbgapi\_address\_space\_get\_info
  - Memory, [78](#)
- amd\_dbgapi\_address\_space\_id\_t, [103](#)
  - handle, [104](#)
- amd\_dbgapi\_address\_space\_info\_t
  - Memory, [76](#)
- amd\_dbgapi\_address\_spaces\_may\_alias
  - Memory, [79](#)
- amd\_dbgapi\_agent\_get\_info
  - Agents, [29](#)
- amd\_dbgapi\_agent\_id\_t, [104](#)
  - handle, [104](#)
- amd\_dbgapi\_agent\_info\_t
  - Agents, [29](#)
- amd\_dbgapi\_agent\_list
  - Agents, [30](#)
- amd\_dbgapi\_architecture\_address\_class\_get\_info
  - Memory, [80](#)
- amd\_dbgapi\_architecture\_address\_class\_list
  - Memory, [81](#)
- amd\_dbgapi\_architecture\_address\_space\_list
  - Memory, [81](#)
- amd\_dbgapi\_architecture\_get\_info
  - Architectures, [16](#)
- amd\_dbgapi\_architecture\_id\_t, [105](#)
  - handle, [105](#)
- amd\_dbgapi\_architecture\_info\_t
  - Architectures, [15](#)
- amd\_dbgapi\_architecture\_register\_class\_get\_info
  - Registers, [63](#)
- amd\_dbgapi\_architecture\_register\_class\_list
  - Registers, [64](#)
- amd\_dbgapi\_architecture\_register\_get\_info
  - Registers, [65](#)
- amd\_dbgapi\_architecture\_register\_list
  - Registers, [66](#)
- amd\_dbgapi\_breakpoint\_action\_t
  - Callbacks, [99](#)
- amd\_dbgapi\_breakpoint\_id\_t, [105](#)
  - handle, [105](#)
- amd\_dbgapi\_breakpoint\_state\_t
  - Callbacks, [99](#)
- amd\_dbgapi\_callbacks\_s, [106](#)
  - add\_breakpoint, [107](#)
  - allocate\_memory, [107](#)
  - deallocate\_memory, [107](#)
  - disable\_notify\_shared\_library, [107](#)
  - enable\_notify\_shared\_library, [108](#)
  - get\_os\_pid, [108](#)
  - get\_symbol\_address, [109](#)
  - log\_message, [109](#)
  - remove\_breakpoint, [109](#)
  - set\_breakpoint\_state, [110](#)
- amd\_dbgapi\_callbacks\_t
  - Callbacks, [99](#)
- amd\_dbgapi\_changed\_t
  - Basic Types, [6](#)
- amd\_dbgapi\_client\_process\_id\_t
  - Processes, [20](#)
- amd\_dbgapi\_client\_thread\_id\_t
  - Callbacks, [99](#)
- amd\_dbgapi\_code\_object\_get\_info
  - Code Objects, [25](#)
- amd\_dbgapi\_code\_object\_id\_t, [110](#)
  - handle, [111](#)
- amd\_dbgapi\_code\_object\_info\_t
  - Code Objects, [25](#)
- amd\_dbgapi\_code\_object\_list
  - Code Objects, [26](#)
- amd\_dbgapi\_convert\_address\_space
  - Memory, [82](#)
- amd\_dbgapi\_disassemble\_instruction
  - Architectures, [16](#)
- amd\_dbgapi\_dispatch\_barrier\_t
  - Dispatches, [40](#)
- amd\_dbgapi\_dispatch\_fence\_scope\_t
  - Dispatches, [40](#)
- amd\_dbgapi\_dispatch\_get\_info
  - Dispatches, [41](#)
- amd\_dbgapi\_dispatch\_id\_t, [111](#)
  - handle, [111](#)
- amd\_dbgapi\_dispatch\_info\_t
  - Dispatches, [40](#)
- amd\_dbgapi\_dispatch\_list
  - Dispatches, [42](#)
- amd\_dbgapi\_displaced\_stepping\_complete
  - Displaced Stepping, [53](#)
- amd\_dbgapi\_displaced\_stepping\_id\_t, [111](#)
  - handle, [112](#)
- amd\_dbgapi\_displaced\_stepping\_start
  - Displaced Stepping, [54](#)
- amd\_dbgapi\_dwarf\_address\_class\_to\_address\_class
  - Memory, [83](#)
- amd\_dbgapi\_dwarf\_address\_space\_to\_address\_space
  - Memory, [84](#)
- amd\_dbgapi\_dwarf\_register\_to\_register
  - Registers, [67](#)
- amd\_dbgapi\_event\_get\_info
  - Events, [92](#)

- amd\_dbgapi\_event\_id\_t, [112](#)
  - handle, [112](#)
- amd\_dbgapi\_event\_info\_t
  - Events, [90](#)
- amd\_dbgapi\_event\_kind\_t
  - Events, [90](#)
- amd\_dbgapi\_event\_processed
  - Events, [93](#)
- amd\_dbgapi\_finalize
  - Initialization and Finalization, [12](#)
- amd\_dbgapi\_get\_architecture
  - Architectures, [17](#)
- amd\_dbgapi\_get\_build\_name
  - Versioning, [11](#)
- amd\_dbgapi\_get\_status\_string
  - Status Codes, [9](#)
- amd\_dbgapi\_get\_version
  - Versioning, [11](#)
- amd\_dbgapi\_global\_address\_t
  - Basic Types, [5](#)
- amd\_dbgapi\_initialize
  - Initialization and Finalization, [12](#)
- amd\_dbgapi\_lane\_id\_t
  - Memory, [75](#)
- amd\_dbgapi\_log\_level\_t
  - Logging, [95](#)
- amd\_dbgapi\_memory\_precision\_t
  - Memory, [77](#)
- amd\_dbgapi\_next\_pending\_event
  - Events, [93](#)
- amd\_dbgapi\_notifier\_t
  - Basic Types, [5](#)
- amd\_dbgapi\_os\_pid
  - Basic Types, [6](#)
- amd\_dbgapi\_prefetch\_register
  - Registers, [67](#)
- amd\_dbgapi\_process\_attach
  - Processes, [21](#)
- amd\_dbgapi\_process\_detach
  - Processes, [22](#)
- amd\_dbgapi\_process\_get\_info
  - Processes, [22](#)
- amd\_dbgapi\_process\_id\_t, [112](#)
  - handle, [113](#)
- amd\_dbgapi\_process\_info\_t
  - Processes, [20](#)
- amd\_dbgapi\_process\_set\_progress
  - Processes, [23](#)
- amd\_dbgapi\_progress\_t
  - Processes, [20](#)
- amd\_dbgapi\_queue\_error\_reason\_t
  - Queues, [33](#)
- amd\_dbgapi\_queue\_get\_info
  - Queues, [34](#)
- amd\_dbgapi\_queue\_id\_t, [113](#)
  - handle, [113](#)
- amd\_dbgapi\_queue\_info\_t
  - Queues, [33](#)
- amd\_dbgapi\_queue\_list
  - Queues, [36](#)
- amd\_dbgapi\_queue\_packet\_id\_t
  - Queues, [33](#)
- amd\_dbgapi\_queue\_packet\_list
  - Queues, [37](#)
- amd\_dbgapi\_queue\_state\_t
  - Queues, [34](#)
- amd\_dbgapi\_queue\_type\_t
  - Queues, [34](#)
- amd\_dbgapi\_read\_memory
  - Memory, [85](#)
- amd\_dbgapi\_read\_register
  - Registers, [68](#)
- amd\_dbgapi\_register\_class\_id\_t, [113](#)
  - handle, [114](#)
- amd\_dbgapi\_register\_class\_info\_t
  - Registers, [62](#)
- amd\_dbgapi\_register\_class\_state\_t
  - Registers, [63](#)
- amd\_dbgapi\_register\_id\_t, [114](#)
  - handle, [114](#)
- amd\_dbgapi\_register\_info\_t
  - Registers, [63](#)
- amd\_dbgapi\_register\_is\_in\_register\_class
  - Registers, [69](#)
- amd\_dbgapi\_remove\_watchpoint
  - Watchpoints, [58](#)
- amd\_dbgapi\_report\_breakpoint\_hit
  - Callbacks, [100](#)
- amd\_dbgapi\_report\_shared\_library
  - Callbacks, [101](#)
- amd\_dbgapi\_resume\_mode\_t
  - Wave, [44](#)
- amd\_dbgapi\_runtime\_state\_t
  - Events, [91](#)
- amd\_dbgapi\_segment\_address\_t
  - Memory, [75](#)
- amd\_dbgapi\_set\_log\_level
  - Logging, [95](#)
- amd\_dbgapi\_set\_memory\_precision
  - Memory, [86](#)
- amd\_dbgapi\_set\_watchpoint
  - Watchpoints, [59](#)
- amd\_dbgapi\_shared\_library\_id\_t, [115](#)
  - handle, [115](#)
- amd\_dbgapi\_shared\_library\_state\_t
  - Callbacks, [100](#)
- amd\_dbgapi\_size\_t
  - Basic Types, [6](#)

- amd\_dbgapi\_status\_t
  - Status Codes, [8](#)
- amd\_dbgapi\_watchpoint\_id\_t
  - Watchpoints, [57](#)
- amd\_dbgapi\_watchpoint\_kind\_t
  - Watchpoints, [57](#)
- amd\_dbgapi\_watchpoint\_share\_kind\_t
  - Watchpoints, [57](#)
- amd\_dbgapi\_wave\_get\_info
  - Wave, [47](#)
- amd\_dbgapi\_wave\_id\_t, [115](#)
  - handle, [115](#)
- amd\_dbgapi\_wave\_info\_t
  - Wave, [44](#)
- amd\_dbgapi\_wave\_list
  - Wave, [48](#)
- amd\_dbgapi\_wave\_register\_get\_info
  - Registers, [70](#)
- amd\_dbgapi\_wave\_register\_list
  - Registers, [71](#)
- amd\_dbgapi\_wave\_resume
  - Wave, [49](#)
- amd\_dbgapi\_wave\_state\_t
  - Wave, [45](#)
- amd\_dbgapi\_wave\_stop
  - Wave, [50](#)
- amd\_dbgapi\_wave\_stop\_reason\_t
  - Wave, [45](#)
- amd\_dbgapi\_write\_memory
  - Memory, [87](#)
- amd\_dbgapi\_write\_register
  - Registers, [71](#)
- Architectures, [14](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION\_PC\_ADJUST, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_BREAKPOINT\_INSTRUCTION\_SIZE, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_DEFAULT\_GLOBAL\_ADDRESS\_SPACE, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_ELF\_AMDGPU\_MACHINE, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_EXECUTION\_MASK\_REGISTER, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_LARGEST\_INSTRUCTION\_SIZE, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_MINIMUM\_INSTRUCTION\_ALIGNMENT, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_NAME, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_PC\_REGISTER, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_PRECISE\_MEMORY\_SUPPORTED, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_WATCHPOINT\_COUNT, [15](#)
  - AMD\_DBGAPI\_ARCHITECTURE\_INFO\_WATCHPOINT\_SHARE, [15](#)
  - amd\_dbgapi\_architecture\_get\_info, [16](#)
  - amd\_dbgapi\_architecture\_info\_t, [15](#)
  - amd\_dbgapi\_disassemble\_instruction, [16](#)
  - amd\_dbgapi\_get\_architecture, [17](#)
- Basic Types, [5](#)
  - AMD\_DBGAPI\_CHANGED\_NO, [6](#)
  - AMD\_DBGAPI\_CHANGED\_YES, [6](#)
  - amd\_dbgapi\_changed\_t, [6](#)
  - amd\_dbgapi\_global\_address\_t, [5](#)
  - amd\_dbgapi\_notifier\_t, [5](#)
  - amd\_dbgapi\_os\_pid, [6](#)
  - amd\_dbgapi\_size\_t, [6](#)
- Callbacks, [98](#)
  - AMD\_DBGAPI\_BREAKPOINT\_ACTION\_HALT, [99](#)
  - AMD\_DBGAPI\_BREAKPOINT\_ACTION\_RESUME, [99](#)
  - AMD\_DBGAPI\_BREAKPOINT\_STATE\_DISABLE, [100](#)
  - AMD\_DBGAPI\_BREAKPOINT\_STATE\_ENABLE, [100](#)
  - AMD\_DBGAPI\_SHARED\_LIBRARY\_STATE\_LOADED, [100](#)
  - AMD\_DBGAPI\_SHARED\_LIBRARY\_STATE\_UNLOADED, [100](#)
  - amd\_dbgapi\_breakpoint\_action\_t, [99](#)
  - amd\_dbgapi\_breakpoint\_state\_t, [99](#)
  - amd\_dbgapi\_callbacks\_t, [99](#)
  - amd\_dbgapi\_client\_thread\_id\_t, [99](#)
  - amd\_dbgapi\_report\_breakpoint\_hit, [100](#)
  - amd\_dbgapi\_report\_shared\_library, [101](#)
  - amd\_dbgapi\_shared\_library\_state\_t, [100](#)
- Code Objects, [24](#)
  - AMD\_DBGAPI\_CODE\_OBJECT\_INFO\_LOAD\_ADDRESS, [25](#)
  - AMD\_DBGAPI\_CODE\_OBJECT\_INFO\_URI\_NAME, [25](#)
  - amd\_dbgapi\_code\_object\_get\_info, [25](#)
  - amd\_dbgapi\_code\_object\_info\_t, [25](#)
  - amd\_dbgapi\_code\_object\_list, [26](#)
- deallocate\_memory
  - amd\_dbgapi\_callbacks\_s, [107](#)
- disable\_notify\_shared\_library
  - amd\_dbgapi\_callbacks\_s, [107](#)
- Dispatches, [39](#)
  - AMD\_DBGAPI\_DISPATCH\_BARRIER\_NONE, [40](#)
  - AMD\_DBGAPI\_DISPATCH\_BARRIER\_PRESENT, [40](#)

- AMD\_DBGAPI\_DISPATCH\_FENCE\_SCOPE\_AGENT, 40
- AMD\_DBGAPI\_DISPATCH\_FENCE\_SCOPE\_NONE, 40
- AMD\_DBGAPI\_DISPATCH\_FENCE\_SCOPE\_SYSTEM, 40
- AMD\_DBGAPI\_DISPATCH\_INFO\_ACQUIRE\_FENCE, 40
- AMD\_DBGAPI\_DISPATCH\_INFO\_AGENT, 40
- AMD\_DBGAPI\_DISPATCH\_INFO\_ARCHITECTURE, 40
- AMD\_DBGAPI\_DISPATCH\_INFO\_BARRIER, 40
- AMD\_DBGAPI\_DISPATCH\_INFO\_GRID\_DIMENSIONS, 41
- AMD\_DBGAPI\_DISPATCH\_INFO\_GRID\_SIZES, 41
- AMD\_DBGAPI\_DISPATCH\_INFO\_GROUP\_SEGMENT\_SIZE, 41
- AMD\_DBGAPI\_DISPATCH\_INFO\_KERNEL\_ARGUMENT\_SEGMENT\_ADDRESS, 41
- AMD\_DBGAPI\_DISPATCH\_INFO\_KERNEL\_ENTRY\_ADDRESS, 41
- AMD\_DBGAPI\_DISPATCH\_INFO\_PACKET\_ID, 40
- AMD\_DBGAPI\_DISPATCH\_INFO\_PRIVATE\_SEGMENT\_SIZE, 41
- AMD\_DBGAPI\_DISPATCH\_INFO\_QUEUE, 40
- AMD\_DBGAPI\_DISPATCH\_INFO\_RELEASE\_FENCE, 41
- AMD\_DBGAPI\_DISPATCH\_INFO\_WORK\_GROUP\_SIZES, 41
- amd\_dbgapi\_dispatch\_barrier\_t, 40
- amd\_dbgapi\_dispatch\_fence\_scope\_t, 40
- amd\_dbgapi\_dispatch\_get\_info, 41
- amd\_dbgapi\_dispatch\_info\_t, 40
- amd\_dbgapi\_dispatch\_list, 42
- Displaced Stepping, 52
  - amd\_dbgapi\_displaced\_stepping\_complete, 53
  - amd\_dbgapi\_displaced\_stepping\_start, 54
- enable\_notify\_shared\_library
  - amd\_dbgapi\_callbacks\_s, 108
- Events, 89
  - AMD\_DBGAPI\_EVENT\_INFO\_BREAKPOINT, 90
  - AMD\_DBGAPI\_EVENT\_INFO\_CLIENT\_THREAD, 90
  - AMD\_DBGAPI\_EVENT\_INFO\_KIND, 90
  - AMD\_DBGAPI\_EVENT\_INFO\_RUNTIME\_STATE, 90
  - AMD\_DBGAPI\_EVENT\_INFO\_RUNTIME\_VERSION, 90
  - AMD\_DBGAPI\_EVENT\_INFO\_WAVE, 90
  - AMD\_DBGAPI\_EVENT\_KIND\_BREAKPOINT\_RESULT, 91
  - AMD\_DBGAPI\_EVENT\_KIND\_CODE\_OBJECT\_LIST\_UPDATED, 91
  - AMD\_DBGAPI\_EVENT\_KIND\_NONE, 91
  - AMD\_DBGAPI\_EVENT\_KIND\_QUEUE\_ERROR, 91
  - AMD\_DBGAPI\_EVENT\_KIND\_RUNTIME, 91
  - AMD\_DBGAPI\_EVENT\_KIND\_WAVE\_COMMAND\_TERMINATED, 91
  - AMD\_DBGAPI\_EVENT\_KIND\_WAVE\_STOP, 91
  - AMD\_DBGAPI\_RUNTIME\_STATE\_LOADED\_SUPPORTED, 92
  - AMD\_DBGAPI\_RUNTIME\_STATE\_LOADED\_UNSUPPORTED, 92
  - AMD\_DBGAPI\_RUNTIME\_STATE\_UNLOADED, 92
  - amd\_dbgapi\_event\_get\_info, 92
  - amd\_dbgapi\_event\_info\_t, 90
  - amd\_dbgapi\_event\_kind\_t, 90
  - amd\_dbgapi\_event\_processed, 93
  - amd\_dbgapi\_next\_pending\_event, 93
  - amd\_dbgapi\_runtime\_state\_t, 91
- get\_os\_pid
  - amd\_dbgapi\_callbacks\_s, 108
- get\_symbol\_address
  - amd\_dbgapi\_callbacks\_s, 109
- handle
  - amd\_dbgapi\_address\_class\_id\_t, 103
  - amd\_dbgapi\_address\_space\_id\_t, 104
  - amd\_dbgapi\_agent\_id\_t, 104
  - amd\_dbgapi\_architecture\_id\_t, 105
  - amd\_dbgapi\_breakpoint\_id\_t, 105
  - amd\_dbgapi\_code\_object\_id\_t, 111
  - amd\_dbgapi\_dispatch\_id\_t, 111
  - amd\_dbgapi\_displaced\_stepping\_id\_t, 112
  - amd\_dbgapi\_event\_id\_t, 112
  - amd\_dbgapi\_process\_id\_t, 113
  - amd\_dbgapi\_queue\_id\_t, 113
  - amd\_dbgapi\_register\_class\_id\_t, 114
  - amd\_dbgapi\_register\_id\_t, 114
  - amd\_dbgapi\_shared\_library\_id\_t, 115
  - amd\_dbgapi\_wave\_id\_t, 115
- include/amd-dbgapi.h, 117
- Initialization and Finalization, 12
  - amd\_dbgapi\_finalize, 12
  - amd\_dbgapi\_initialize, 12
- log\_message
  - amd\_dbgapi\_callbacks\_s, 109
- Logging, 95
  - AMD\_DBGAPI\_LOG\_LEVEL\_FATAL\_ERROR, 95
  - AMD\_DBGAPI\_LOG\_LEVEL\_INFO, 95
  - AMD\_DBGAPI\_LOG\_LEVEL\_NONE, 95
  - AMD\_DBGAPI\_LOG\_LEVEL\_VERBOSE, 95
  - AMD\_DBGAPI\_LOG\_LEVEL\_WARNING, 95
  - amd\_dbgapi\_log\_level\_t, 95
  - amd\_dbgapi\_set\_log\_level, 95

## Memory, 73

AMD\_DBGAPI\_ADDRESS\_CLASS\_INFO\_ADDRESS\_SPACE, 76  
 AMD\_DBGAPI\_ADDRESS\_CLASS\_INFO\_NAME, 76  
 AMD\_DBGAPI\_ADDRESS\_CLASS\_STATE\_MEMBER, 76  
 AMD\_DBGAPI\_ADDRESS\_CLASS\_STATE\_NOT\_MEMBER, 76  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_ALL, 76  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_DISPATCH\_CONSTANT, 76  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_ACCESS\_PROGRAM\_CONSTANT, 76  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_ALIAS\_MAY, 76  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_ALIAS\_NONE, 76  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_ACCESS, 77  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_ADDRESS\_SIZE, 77  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_NAME, 77  
 AMD\_DBGAPI\_ADDRESS\_SPACE\_INFO\_NULL\_ADDRESS, 77  
 AMD\_DBGAPI\_MEMORY\_PRECISION\_NONE, 77  
 AMD\_DBGAPI\_MEMORY\_PRECISION\_PRECISE, 77  
 amd\_dbgapi\_address\_class\_info\_t, 75  
 amd\_dbgapi\_address\_class\_state\_t, 76  
 amd\_dbgapi\_address\_is\_in\_address\_class, 77  
 amd\_dbgapi\_address\_space\_access\_t, 76  
 amd\_dbgapi\_address\_space\_alias\_t, 76  
 amd\_dbgapi\_address\_space\_get\_info, 78  
 amd\_dbgapi\_address\_space\_info\_t, 76  
 amd\_dbgapi\_address\_spaces\_may\_alias, 79  
 amd\_dbgapi\_architecture\_address\_class\_get\_info, 80  
 amd\_dbgapi\_architecture\_address\_class\_list, 81  
 amd\_dbgapi\_architecture\_address\_space\_list, 81  
 amd\_dbgapi\_convert\_address\_space, 82  
 amd\_dbgapi\_dwarf\_address\_class\_to\_address\_class, 83  
 amd\_dbgapi\_dwarf\_address\_space\_to\_address\_space, 84  
 amd\_dbgapi\_lane\_id\_t, 75  
 amd\_dbgapi\_memory\_precision\_t, 77  
 amd\_dbgapi\_read\_memory, 85  
 amd\_dbgapi\_segment\_address\_t, 75  
 amd\_dbgapi\_set\_memory\_precision, 86  
 amd\_dbgapi\_write\_memory, 87

## Processes, 19

AMD\_DBGAPI\_PROCESS\_INFO\_NOTIFIER, 20  
 AMD\_DBGAPI\_PROGRESS\_NO\_FORWARD, 20  
 AMD\_DBGAPI\_PROGRESS\_NORMAL, 20  
 amd\_dbgapi\_client\_process\_id\_t, 20  
 amd\_dbgapi\_process\_attach, 21  
 amd\_dbgapi\_process\_detach, 22  
 amd\_dbgapi\_process\_get\_info, 22  
 amd\_dbgapi\_process\_info\_t, 20  
 amd\_dbgapi\_process\_set\_progress, 23  
 amd\_dbgapi\_progress\_t, 20

## Queues, 32

AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_ASSERT\_TRAP, 33  
 AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_INVALID\_PACKET, 33  
 AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_MEMORY\_VIOLATION, 33  
 AMD\_DBGAPI\_QUEUE\_ERROR\_REASON\_WAVE\_ERROR, 33  
 AMD\_DBGAPI\_QUEUE\_INFO\_AGENT, 33  
 AMD\_DBGAPI\_QUEUE\_INFO\_ARCHITECTURE, 33  
 AMD\_DBGAPI\_QUEUE\_INFO\_ERROR\_REASON, 34  
 AMD\_DBGAPI\_QUEUE\_INFO\_STATE, 34  
 AMD\_DBGAPI\_QUEUE\_STATE\_ERROR, 34  
 AMD\_DBGAPI\_QUEUE\_STATE\_VALID, 34  
 AMD\_DBGAPI\_QUEUE\_TYPE, 34  
 AMD\_DBGAPI\_QUEUE\_TYPE\_AMD\_PM4, 34  
 AMD\_DBGAPI\_QUEUE\_TYPE\_HSA\_KERNEL\_DISPATCH\_MULTIPLE\_PRODUCER, 34  
 AMD\_DBGAPI\_QUEUE\_TYPE\_HSA\_KERNEL\_DISPATCH\_SINGLE\_PRODUCER, 34  
 AMD\_DBGAPI\_QUEUE\_TYPE\_UNKNOWN, 34  
 amd\_dbgapi\_queue\_error\_reason\_t, 33  
 amd\_dbgapi\_queue\_get\_info, 34  
 amd\_dbgapi\_queue\_info\_t, 33  
 amd\_dbgapi\_queue\_list, 36  
 amd\_dbgapi\_queue\_packet\_id\_t, 33  
 amd\_dbgapi\_queue\_packet\_list, 37  
 amd\_dbgapi\_queue\_state\_t, 34  
 amd\_dbgapi\_queue\_type\_t, 34

## Registers, 61

AMD\_DBGAPI\_REGISTER\_CLASS\_INFO\_NAME, 63  
 AMD\_DBGAPI\_REGISTER\_CLASS\_STATE\_MEMBER, 63  
 AMD\_DBGAPI\_REGISTER\_CLASS\_STATE\_NOT\_MEMBER, 63  
 AMD\_DBGAPI\_REGISTER\_INFO\_NAME, 63  
 AMD\_DBGAPI\_REGISTER\_INFO\_SIZE, 63  
 AMD\_DBGAPI\_REGISTER\_INFO\_TYPE, 63



- amd\_dbgapi\_architecture\_register\_class\_get\_info, 63
- amd\_dbgapi\_architecture\_register\_class\_list, 64
- amd\_dbgapi\_architecture\_register\_get\_info, 65
- amd\_dbgapi\_architecture\_register\_list, 66
- amd\_dbgapi\_dwarf\_register\_to\_register, 67
- amd\_dbgapi\_prefetch\_register, 67
- amd\_dbgapi\_read\_register, 68
- amd\_dbgapi\_register\_class\_info\_t, 62
- amd\_dbgapi\_register\_class\_state\_t, 63
- amd\_dbgapi\_register\_info\_t, 63
- amd\_dbgapi\_register\_is\_in\_register\_class, 69
- amd\_dbgapi\_wave\_register\_get\_info, 70
- amd\_dbgapi\_wave\_register\_list, 71
- amd\_dbgapi\_write\_register, 71
- remove\_breakpoint
  - amd\_dbgapi\_callbacks\_s, 109
- set\_breakpoint\_state
  - amd\_dbgapi\_callbacks\_s, 110
- Status Codes, 7
  - AMD\_DBGAPI\_STATUS\_ERROR, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_ALREADY\_ATTACHED, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_ALREADY\_INITIALIZED, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_CLIENT\_CALLBACK, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_DISPLACED\_STEPPING\_BUFFER\_UNAVAILABLE, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_ILLEGAL\_INSTRUCTION, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_CLASS\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_SPACE\_CONVERSION, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ADDRESS\_SPACE\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_AGENT\_ID, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARCHITECTURE\_ID, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARGUMENT, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ARGUMENT\_SIZE, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_BREAKPOINT\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_CLIENT\_PROCESS\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_CODE\_OBJECT\_ID, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_DISPATCH\_ID, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_DISPLACED\_STEPPING\_ID, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_ELF\_AMDGPU\_MACHINE, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_EVENT\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_LANE\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_PROCESS\_ID, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_QUEUE\_ID, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_REGISTER\_CLASS\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_REGISTER\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_SHARED\_LIBRARY\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_WATCHPOINT\_ID, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_INVALID\_WAVE\_ID, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_LIBRARY\_NOT\_LOADED, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_MEMORY\_ACCESS, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_NO\_WATCHPOINT\_AVAILABLE, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_NOT\_INITIALIZED, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_NOT\_SUPPORTED, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_PROCESS\_EXITED, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_SYMBOL\_NOT\_FOUND, 9
  - AMD\_DBGAPI\_STATUS\_ERROR\_VERSION\_MISMATCH, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_NOT\_RESUMABLE, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_NOT\_STOPPED, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_OUTSTANDING\_STOP, 8
  - AMD\_DBGAPI\_STATUS\_ERROR\_WAVE\_STOPPED, 8
  - AMD\_DBGAPI\_STATUS\_FATAL, 8
  - AMD\_DBGAPI\_STATUS\_SUCCESS, 8
  - amd\_dbgapi\_get\_status\_string, 9
  - amd\_dbgapi\_status\_t, 8
- Versioning, 10

AMD\_DBGAPI\_VERSION\_MAJOR, 10  
 AMD\_DBGAPI\_VERSION\_MINOR, 10  
 amd\_dbgapi\_get\_build\_name, 11  
 amd\_dbgapi\_get\_version, 11

#### Watchpoints, 56

AMD\_DBGAPI\_WATCHPOINT\_KIND\_ALL, 57  
 AMD\_DBGAPI\_WATCHPOINT\_KIND\_LOAD, 57  
 AMD\_DBGAPI\_WATCHPOINT\_KIND\_RMW, 57  
 AMD\_DBGAPI\_WATCHPOINT\_KIND\_STORE\_AND\_RMW, 57  
 AMD\_DBGAPI\_WATCHPOINT\_SHARE\_KIND\_SHARED, 57  
 AMD\_DBGAPI\_WATCHPOINT\_SHARE\_KIND\_UNSHARED, 57  
 AMD\_DBGAPI\_WATCHPOINT\_SHARE\_KIND\_UNSUPPORTED, 57  
 amd\_dbgapi\_remove\_watchpoint, 58  
 amd\_dbgapi\_set\_watchpoint, 59  
 amd\_dbgapi\_watchpoint\_id\_t, 57  
 amd\_dbgapi\_watchpoint\_kind\_t, 57  
 amd\_dbgapi\_watchpoint\_share\_kind\_t, 57

#### Wave, 43

AMD\_DBGAPI\_RESUME\_MODE\_NORMAL, 44  
 AMD\_DBGAPI\_RESUME\_MODE\_SINGLE\_STEP, 44  
 AMD\_DBGAPI\_WAVE\_INFO\_AGENT, 45  
 AMD\_DBGAPI\_WAVE\_INFO\_ARCHITECTURE, 45  
 AMD\_DBGAPI\_WAVE\_INFO\_DISPATCH, 45  
 AMD\_DBGAPI\_WAVE\_INFO\_EXEC\_MASK, 45  
 AMD\_DBGAPI\_WAVE\_INFO\_LANE\_COUNT, 45  
 AMD\_DBGAPI\_WAVE\_INFO\_PC, 45  
 AMD\_DBGAPI\_WAVE\_INFO\_QUEUE, 45  
 AMD\_DBGAPI\_WAVE\_INFO\_STATE, 44  
 AMD\_DBGAPI\_WAVE\_INFO\_STOP\_REASON, 44  
 AMD\_DBGAPI\_WAVE\_INFO\_WATCHPOINTS, 44  
 AMD\_DBGAPI\_WAVE\_INFO\_WAVE\_NUMBER\_IN\_WORK\_GROUP, 45  
 AMD\_DBGAPI\_WAVE\_INFO\_WORK\_GROUP\_COORD, 45  
 AMD\_DBGAPI\_WAVE\_STATE\_RUN, 45  
 AMD\_DBGAPI\_WAVE\_STATE\_SINGLE\_STEP, 45  
 AMD\_DBGAPI\_WAVE\_STATE\_STOP, 45  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ASSERT\_TRAP, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_BREAKPOINT, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_DEBUG\_TRAP, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ECC\_ERROR, 47  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FATAL\_HALT, 47

AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_DIVIDE\_BY\_0, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INEXACT, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INPUT\_DENORMAL, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_INVALID\_OPERATION, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_OVERFLOW, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_FP\_UNDERFLOW, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_ILLEGAL\_INSTRUCTION, 47  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_INT\_DIVIDE\_BY\_0, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_MEMORY\_VIOLATION, 47  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_NONE, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_QUEUE\_ERROR, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_SINGLE\_STEP, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_TRAP, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_WATCHPOINT, 46  
 AMD\_DBGAPI\_WAVE\_STOP\_REASON\_XNACK\_ERROR, 47  
 amd\_dbgapi\_resume\_mode\_t, 44  
 amd\_dbgapi\_wave\_get\_info, 47  
 amd\_dbgapi\_wave\_info\_t, 44  
 amd\_dbgapi\_wave\_list, 48  
 amd\_dbgapi\_wave\_resume, 49  
 amd\_dbgapi\_wave\_state\_t, 45  
 amd\_dbgapi\_wave\_stop, 50  
 amd\_dbgapi\_wave\_stop\_reason\_t, 45