AMD_DBGAPI

0.20.2

Generated by Doxygen 1.8.11

Fri May 8 2020 14:33:46

Contents

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
2. AMD ROCm Platform: rocm-documentation.readthedocs.io/en/latest
3. Bus:Device.Function (BDF) Notation: wiki.xen.org/wiki/Bus:Device.Function_(BDF)_Notation
4. HSA Platform System Architecture Specification: 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
6. Semantic Versioning: semver.org
7. The LLVM Compiler Infrastructure: llvm.org
8. User Guide for AMDGPU LLVM Backend: llvm.org/docs/AMDGPUUsage.html

Chapter 2

Module Documentation

2.1 Symbol Versions

The names used for the shared library versioned symbols.

Macros

- #define AMD_DBGAPI_VERSION_0_1

  The function was introduced in version 0.1 of the interface and has the symbol version string of "AMD_DBGAPI_0.1".
- #define AMD_DBGAPI_VERSION_0_20

  The function was introduced in version 0.20 of the interface and has the symbol version string of "AMD_DBGAPI_0.20".

2.1.1 Detailed Description

The names used for the shared library versioned symbols.

Every function is annotated with one of the version macros defined in this section. Each macro specifies a corresponding symbol version string. After dynamically loading the shared library with `dlopen`, the address of each function can be obtained using `dlvsym` with the name of the function and its corresponding symbol version string. An error will be reported by `dlvsym` if the installed library does not support the version for the function specified in this version of the interface.

2.1.2 Macro Definition Documentation

2.1.2.1 #define AMD_DBGAPI_VERSION_0_1

The function was introduced in version 0.1 of the interface and has the symbol version string of "AMD_DBGAPI_0.1".

2.1.2.2 #define AMD_DBGAPI_VERSION_0_20

The function was introduced in version 0.20 of the interface and has the symbol version string of "AMD_DBGAPI_↩0.20".

2.2 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.

2.2.1 Detailed Description

Types used for common properties.

Note that in some cases enumeration types are used as output parameters for functions using pointers. The C language does not define the underlying type used for enumeration types. This interface requires that the underlying type used by the client will be `int` with a size of 32 bits, and that enumeration types passed by value to functions, or return as values from functions, will have the platform function ABI representation.

2.2.2 Typedef Documentation

2.2.2.1 typedef uint64_t amd_dbgapi_global_address_t

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

2.2.2.2 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® 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.

2.2.2.3 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.

2.2.2.4 typedef uint64_t amd_dbgapi_size_t

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

2.2.3 Enumeration Type Documentation

2.2.3.1 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.

2.3 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_F↩
  ATAL = -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_ST↩
  ATUS_ERROR_NOT_INITIALIZED = -7,
  AMD_DBGAPI_STATUS_ERROR_VERSION_MISMATCH = -8, AMD_DBGAPI_STATUS_ERROR_ALREAD↩
  Y_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_I↩
  NVALID_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_OUT↩
  STANDING_STOP = -21, AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_RESUMABLE = -22, AMD_DBGA↩
  PI_STATUS_ERROR_INVALID_DISPLACED_STEPPING_ID = -23,
  AMD_DBGAPI_STATUS_ERROR_DISPLACED_STEPPING_BUFFER_UNAVAILABLE = -24, AMD_DBGAP↩
  I_STATUS_ERROR_INVALID_WATCHPOINT_ID = -25, AMD_DBGAPI_STATUS_ERROR_NO_WATCHPOI↩
  NT_AVAILABLE = -26, AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_CLASS_ID = -27,
  AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID = -28, AMD_DBGAPI_STATUS_ERROR_INVAL↩
  ID_LANE_ID = -29, AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_CLASS_ID = -30, AMD_DBGAP↩
  I_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, A↩
  MD_DBGAPI_STATUS_ERROR_INVALID_SHARED_LIBRARY_ID = -35,
  AMD_DBGAPI_STATUS_ERROR_INVALID_BREAKPOINT_ID = -36, AMD_DBGAPI_STATUS_ERROR_CLI↩
  ENT_CALLBACK = -37, AMD_DBGAPI_STATUS_ERROR_INVALID_CLIENT_PROCESS_ID = -38, AMD_D↩
  BGAPI_STATUS_ERROR_PROCESS_EXITED = -39,
  AMD_DBGAPI_STATUS_ERROR_LIBRARY_NOT_LOADED = -40, AMD_DBGAPI_STATUS_ERROR_SYM↩
  BOL_NOT_FOUND = -41, AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS = -42, AMD_DBGAPI_ST↩
  ATUS_ERROR_UNIMPLEMENTED = INT32_MIN }

  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) AMD_DBGAPI_VERSION_0_1

  Query a textual description of a status code.

2.3.1 Detailed Description

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

2.3.2 Enumeration Type Documentation

2.3.2.1 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.

AMD_DBGAPI_STATUS_ERROR_UNIMPLEMENTED The operation is not currently implemented. This error may be reported by any function. Check the `README.md` file of the library implementation to determine the status of its implementation of the interface.

2.3.3 Function Documentation

2.3.3.1 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	*status*	Status code.
out	*status_string*	A NUL terminated string that describes the status code. The string is read only and owned by the library.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully. `status_string` has been updated.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`status` is an invalid status code or `status_string` is NULL. `status_string` is unaltered.

2.4 Versioning

Version information about the interface and the associated installed library.

Enumerations

- enum { AMD_DBGAPI_VERSION_MAJOR = 0, AMD_DBGAPI_VERSION_MINOR = 20 }
  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) AMD_DBGAPI_VERSION_0_1
  Query the version of the installed library.
- const char AMD_DBGAPI * amd_dbgapi_get_build_name (void) AMD_DBGAPI_VERSION_0_1
  Query the installed library build name.

2.4.1 Detailed Description

Version information about the interface and the associated installed library.

2.4.2 Enumeration Type Documentation

2.4.2.1 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.

2.4.3 Function Documentation

2.4.3.1 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.

2.4.3.2 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	*major*	The major version number is stored if non-NULL.
out	*minor*	The minor version number is stored if non-NULL.
out	*patch*	The patch version number is stored if non-NULL.

2.5 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) AMD_DBG↩ API_VERSION_0_1

  Initialize the library.
- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_finalize (void) AMD_DBGAPI_VERSION_0_1

  Finalize the library.

2.5.1 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.

2.5.2 Function Documentation

2.5.2.1 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

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the library is now uninitialized.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBA↩ CK	This will be reported if any of the amd_dbgapi_callbacks_s callbacks used return an error. The library is still left uninitialized, but the client may be in an inconsistent state.

2.5.2.2 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

in	*callbacks*	A set of callbacks must be provided. These are invoked by certain operations. They are described in amd_dbgapi_callbacks_t.

Return values

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

2.6 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.

Typedefs

- typedef struct amd_dbgapi_symbolizer_id_s * amd_dbgapi_symbolizer_id_t

  Opaque client symbolizer 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_DBGA↩
  PI_ARCHITECTURE_INFO_MINIMUM_INSTRUCTION_ALIGNMENT = 4,
  AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_SIZE = 5, AMD_DBGAPI_ARCHITE↩
  CTURE_INFO_BREAKPOINT_INSTRUCTION = 6, 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_ARCHITECT↩
  URE_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_instruction_kind_t {
  AMD_DBGAPI_INSTRUCTION_KIND_UNKNOWN = 0, AMD_DBGAPI_INSTRUCTION_KIND_SEQUENTIAL =
  1, AMD_DBGAPI_INSTRUCTION_KIND_DIRECT_BRANCH = 2, AMD_DBGAPI_INSTRUCTION_KIND_DIR↩
  ECT_BRANCH_CONDITIONAL = 3,
  AMD_DBGAPI_INSTRUCTION_KIND_INDIRECT_BRANCH_REGISTER_PAIR = 4, AMD_DBGAPI_INSTRU↩
  CTION_KIND_DIRECT_CALL_REGISTER_PAIR = 5, AMD_DBGAPI_INSTRUCTION_KIND_INDIRECT_CA↩
  LL_REGISTER_PAIRS = 6, AMD_DBGAPI_INSTRUCTION_KIND_TERMINATE = 7,
  AMD_DBGAPI_INSTRUCTION_KIND_TRAP = 8, AMD_DBGAPI_INSTRUCTION_KIND_HALT = 9, AMD_DB↩
  GAPI_INSTRUCTION_KIND_BARRIER = 10, AMD_DBGAPI_INSTRUCTION_KIND_SLEEP = 11,
  AMD_DBGAPI_INSTRUCTION_KIND_SPECIAL = 12 }

  The kinds of instruction classifications.

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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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, amd_dbgapi_symbolizer_id_t symbolizer_id, amd_dbgapi_status_t(*symbolizer)(amd_dbgapi_symbolizer_id_t symbolizer_id, amd_dbgapi_global_address_t address, char **symbol_text)) AMD_DBGAPI_VERSION_0_20

  Disassemble a single instruction.

- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_classify_instruction (amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_global_address_t address, amd_dbgapi_size_t *size, const void *memory, amd_dbgapi_instruction_kind_t *instruction_kind, void **instruction_properties) AMD_DBGAPI_VERSION_0_20

  Classify a single instruction.

2.6.1 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.

2.6.2 Macro Definition Documentation

2.6.2.1 #define AMD_DBGAPI_ARCHITECTURE_NONE (amd_dbgapi_architecture_id_t{ 0 })

The NULL architecture handle.

2.6.3 Typedef Documentation

2.6.3.1 typedef struct amd_dbgapi_symbolizer_id_s* amd_dbgapi_symbolizer_id_t

Opaque client symbolizer handle.

A pointer to client data associated with a symbolizer. This pointer is passed to the amd_dbgapi_disassemble_instruction `symbolizer` callback.

2.6.4 Enumeration Type Documentation

2.6.4.1 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_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`. 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.

2.6.4.2 enum amd_dbgapi_instruction_kind_t

The kinds of instruction classifications.

Enumerator

AMD_DBGAPI_INSTRUCTION_KIND_UNKNOWN The instruction classification is unknown. The instruction has no properties.

AMD_DBGAPI_INSTRUCTION_KIND_SEQUENTIAL The instruction executes sequentially. It performs no control flow and the next instruction executed is the following one. The instruction has no properties.

AMD_DBGAPI_INSTRUCTION_KIND_DIRECT_BRANCH The instruction unconditionally branches to a literal address. The instruction properties is of type amd_dbgapi_global_address_t with the value of the target address of the branch.

AMD_DBGAPI_INSTRUCTION_KIND_DIRECT_BRANCH_CONDITIONAL The instruction conditionally branches to a literal address. If the condition is not satisfied then the next instruction is the following one. The instruction properties is of type amd_dbgapi_global_address_t with the value of the target address of the branch if taken.

AMD_DBGAPI_INSTRUCTION_KIND_INDIRECT_BRANCH_REGISTER_PAIR The instruction unconditionally branches to an address held in a pair of registers. The instruction properties is of type amd_dbgapi_register_id_t[2] with the value of the register IDs for the registers. The first register holds the least significant address bits, and the second register holds the most significant address bits.

AMD_DBGAPI_INSTRUCTION_KIND_DIRECT_CALL_REGISTER_PAIR The instruction unconditionally branches to a literal address and the address of the following instruction is saved in a pair of registers. The instruction properties is of type amd_dbgapi_register_id_t[2] with the value of the register IDs for the registers. The register with index 0 holds the least significant address bits, and the register with index 1 holds the most significant address bits.

AMD_DBGAPI_INSTRUCTION_KIND_INDIRECT_CALL_REGISTER_PAIRS The instruction unconditionally branches to an address held in a pair of source registers and the address of the following instruction is saved in a pair of destintion registers. The instruction properties is of type amd_dbgapi_register_id_t[4] with the source register IDs in indicies 0 and 1, and the destination register IDs in indicies 2 and 3. The registers with indicies 0 and 2 hold the least significant address bits, and the registers with indicies 1 and 3 hold the most significant address bits.

AMD_DBGAPI_INSTRUCTION_KIND_TERMINATE The instruction terminates the wave execution. The instruction has no properties.

AMD_DBGAPI_INSTRUCTION_KIND_TRAP The instruction enters the trap handler. The trap handler may return to resume execution, may halt the wave and create an event for amd_dbgapi_next_pending_event to report, or may terminate the wave. The library cannot report execution in the trap handler. If single stepping the trap instruction reports the AMD_DBGAPI_WAVE_STOP_REASON_SINGLE_STEP reason, then the program counter will be at the instruction following the trap instruction, it will not be at the first instruction of the trap handler. It is undefined to set a breakpoint in the trap handler, and will likely cause the inferior to report errors and stop executing correctly. The instruction properties is of type `uint64_t` with the value of the trap code.

AMD_DBGAPI_INSTRUCTION_KIND_HALT The instruction unconditionally halts the wave. The instruction has no properties.

AMD_DBGAPI_INSTRUCTION_KIND_BARRIER The instruction performs some kind of execution barrier which may result in the wave being halted until other waves allow it to continue. Such instructions include wave execution barriers, wave synchronization barriers, and wave semephores. The instruction has no properties.

AMD_DBGAPI_INSTRUCTION_KIND_SLEEP The instruction causes the wave to stop executing for some period of time, before continuing execution with the next instruction. The instruction has no properties.

AMD_DBGAPI_INSTRUCTION_KIND_SPECIAL The instruction has some form of special behavior not covered by any of the other instruction kinds. This likely makes it unsuitable to assume it will execute sequentially. This may include instructions that can affect the execution of other waves waiting at wave synchronization barriers, that may send interrupts, and so forth. The instruction has no properties.

2.6.5 Function Documentation

2.6.5.1 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`	*architecture_id*	The architecture being queried.
`in`	*query*	The query being requested.
`in`	*value_size*	Size of the memory pointed to by `value`. Must be equal to the byte size of the query result.
`out`	*value*	Pointer to memory where the query result is stored.

Return values

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

2.6.5.2 amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_classify_instruction (amd_dbgapi_architecture_id_t *architecture_id,* amd_dbgapi_global_address_t *address,* amd_dbgapi_size_t * *size,* const void * *memory,* amd_dbgapi_instruction_kind_t * *instruction_kind,* void ** *instruction_properties*)

Classify a single instruction.

Parameters

in	*architecture_id*	The architecture to use to perform the classification.
in	*address*	The address of the first byte of the instruction.
in,out	*size*	Pass in the number of bytes available in memory which must be greater than 0. Return the number of bytes consumed to decode the instruction.
in	*memory*	The bytes to decode as an instruction. Must point to an array of at least size bytes. The AMD_DBGAPI_ARCHITECTURE_INFO_LARGEST_INSTRUCTION_SIZE query for architecture_id can be used to determine the number of bytes of the largest instruction. By making size at least this size ensures that the instruction can be decoded if legal. However, size may need to be smaller if no memory exists at the address of address plus size.
out	*instruction_kind*	The classification kind of the instruction.
out	*instruction_properties*	Pointer to the instruction properties that corresponds to the value of instruction_kind. amd_dbgapi_instruction_kind_t defines the type of the instruction properties for each instruction kind value. If the instruction has no properties then NULL is returned. The memory is allocated using the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client. If NULL, no value is returned.

Return values

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

2.6.5.3 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*, amd_dbgapi_symbolizer_id_t *symbolizer_id*, amd_dbgapi_status_t(*)(amd_dbgapi_symbolizer_id_t symbolizer_id, amd_dbgapi_global_address_t address, char **symbol_text) *symbolizer*)

Disassemble a single instruction.

Parameters

in	*architecture_id*	The architecture to use to perform the disassembly.
in	*address*	The address of the first byte of the instruction.
in,out	*size*	Pass in the number of bytes available in `memory` which must be greater than 0. Return the number of bytes consumed to decode the instruction.
in	*memory*	The bytes to decode as an instruction. Must point to an array of at least `size` bytes. The AMD_DBGAPI_ARCHITECTURE_INFO_LARGEST_INSTRUCTION_SIZE query for `architecture_id` can be used to determine the number of bytes of the largest instruction. By making `size` at least this size ensures that the instruction can be decoded if legal. However, `size` may need to be smaller if no memory exists at the address of `address` plus `size`.
out	*instruction_text*	Pointer to NUL terminated string that contains the disassembled textual representation of the instruction. The memory is allocated using the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.
in	*symbolizer_id*	The client handle that is passed to any invocation of the `symbolizer` callback made while disassembling the instruction.
in	*symbolizer*	A callback that is invoked for any operand of the disassembled instruction that is a memory address. It allows the client to provide a symbolic representation of the address as a textual symbol that will be used in the returned `instruction_text`.

If `symbolizer` is NULL, then no symbolization will be performed and any memory addresses will be shown as their numeric address.

If `symbolizer` is non-NULL, the `symbolizer` function will be called with `symbolizer_id` having the value of the above `symbolizer_id` operand, and with `address` having the value of the address of the disassembled instruction's operand.

If the `symbolizer` callback wishes to report a symbol text it must allocate and assign memory for a non-empty NUL terminated `char*` string using a memory allocator that can be deallocated using the amd_dbgapi_callbacks_s::deallocate_memory callback. If must assign the pointer to `symbol_text`, and return AMD_DBGAPI_STATUS_SUCCESS.

If the `symbolizer` callback does not wish to report a symbol it must return AMD_DBGAPI_STATUS_ERROR_SYMBOL_NOT_FOUND.

Any `symbol_text` strings returned by the `symbolizer` callbacks reporting AMD_DBGAPI_STATUS_SUCCESS are deallocated using the amd_dbgapi_callbacks_s::deallocate_memory callback before amd_dbgapi_disassemble_instruction returns.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `size` and `instruction_text`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `size` and `instruction_text` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `size` and `instruction_text` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `size` and `instruction_text` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`size`, `memory`, or `instruction_text` are NULL; or `size` is 0. `size` and `instruction_text` are unaltered.
AMD_DBGAPI_STATUS_ERROR	Encountered an error disassembling the instruction, a `symbolizer` callback returned AMD_DBGAPI_STATUS_SUCCESS with a NULL or empty `symbol_text` string. The bytes may or may not be a legal instruction. `size` and `instruction_text` are unaltered.
AMD_DBGAPI_STATUS_ERROR_ILLEGAL_INSTRUCTION	The bytes starting at `address`, when up to `size` bytes are available, are not a legal instruction for the architecture. `size` and `instruction_text` are unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `instruction_text` returns NULL, or a `symbolizer` callback returns a status other than AMD_DBGAPI_STATUS_SUCCESS and AMD_DBGAPI_STATUS_ERROR_SYMBOL_NOT_FOUND. `size` and `instruction_text` are unaltered.

2.6.5.4 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`	*elf_amdgpu_machine*	The AMD GPU ELF `EF_AMDGPU_MACH` value.
`out`	*architecture_id*	The corresponding architecture.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `architecture_id`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `architecture_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `architecture_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ELF_AMDGPU_MACHINE	`elf_amdgpu_machine` is invalid or unsupported. `architecture_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`architecture_id` is NULL. `architecture_id` is unaltered.

2.7 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 struct amd_dbgapi_client_process_s * 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.
- enum amd_dbgapi_wave_creation_t { AMD_DBGAPI_WAVE_CREATION_NORMAL = 0, AMD_DBGAPI_WAVE_CREATION_STOP = 1 }

  The kinds of wave creation supported by the hardware.

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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Detach from a process and no longer have debug control of the AMD GPU devices it uses.
- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_set_progress (amd_dbgapi_process_id_t process_id, amd_dbgapi_progress_t progress) AMD_DBGAPI_VERSION_0_1

  Set the progress required for a process.
- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_set_wave_creation (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_creation_t creation) AMD_DBGAPI_VERSION_0_20

  Set the wave creation mode for a process.

2.7.1 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.

2.7.2 Macro Definition Documentation

2.7.2.1 #define AMD_DBGAPI_PROCESS_NONE (amd_dbgapi_process_id_t{ 0 })

The NULL process handle.

2.7.3 Typedef Documentation

2.7.3.1 typedef struct amd_dbgapi_client_process_s* 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.

2.7.4 Enumeration Type Documentation

2.7.4.1 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.

2.7.4.2 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. 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.

2.7.4.3 enum amd_dbgapi_wave_creation_t

The kinds of wave creation supported by the hardware.

The hardware creates new waves asynchronously as it executes dispatch packets. If the client requires that all waves are stopped, it needs to first request that the hardware stops creating new waves, followed by halting all already created waves. The wave creation setting allows controlling how the hardware creates new waves for dispatch packets on queues associated with agents belonging to a specific process. It has no affect on waves that have already been created.

Enumerator

AMD_DBGAPI_WAVE_CREATION_NORMAL Normal wave creation allows new waves to be created.

AMD_DBGAPI_WAVE_CREATION_STOP Stop wave creation prevents new waves from being created.

2.7.5 Function Documentation

2.7.5.1 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 proccess 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`	*client_process_id*	The client handle for the process. It is passed as an argument to any callbacks performed to indicate the process being requested.
`out`	*process_id*	The process handle to use for all operations related to this process.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the process is now attached returning `process_id`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `process_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `process_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_ALREADY_ATTACHED	The process is already attached. The process remains attached and `process_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_VERSION_MISMATCH	The installed AMD GPU driver version is not compatible with the library. The process is not attached and `process_id` is unaltered.

Return values

AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`client_process_id` or `process_id` are NULL. The process is not attached and `process_id` is unaltered.

2.7.5.2 amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_detach (amd_dbgapi_process_id_t *process_id*)

Detach from a process and no longer have debug control of the AMD GPU devices it uses.

If the associated native operating system process has already exited, or exits while being detached, then the process is trivially detached.

Otherwise, detaching causes execution of the associated native operating system process to continue unaffected by the library. Any waves with a displaced stepping buffer are stopped and the displaced stepping buffer completed. Any data watchpoints are removed. All agents are configured to AMD_DBGAPI_MEMORY_PRECISION_NONE. Any waves in the stopped or single step state are resumed in non-single step mode. Any pending events are discarded.

After detaching, the process handle becomes invalid. It is undefined to use any handles returned by previous operations performed with a process handle that has become invalid.

A native operating system process can be attached and detached multiple times. Each attach returns a unique process handle even for the same native operating system process.

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

Parameters

process_id	The process handle that is being detached.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the process has been detached from the associated native operating system process, or the associated native operating system process has already exited.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	The `process_id` is invalid. No process is detached.

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

Return values

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

2.7.5.4 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`	*process_id*	The process being controlled.
`in`	*progress*	The progress being set.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the progress has been set.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. The progress setting is not changed.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`progress` is invalid. The progress setting is not changed.

2.7.5.5 amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_set_wave_creation (amd_dbgapi_process_id_t *process_id,* amd_dbgapi_wave_creation_t *creation*)

Set the wave creation mode for a process.

The setting applies to all agents of the specified process.

Parameters

`in`	*process_id*	The process being controlled.
`in`	*creation*	The wave creation mode being set.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the wave creation mode has been set.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. The wave creation mode setting is not changed.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`creation` is invalid. The wave creation setting is not changed.

2.8 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_DBG↩ API_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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Return the list of loaded code objects for a process.

2.8.1 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.

2.8.2 Macro Definition Documentation

2.8.2.1 #define AMD_DBGAPI_CODE_OBJECT_NONE (amd_dbgapi_code_object_id_t{ 0 })

The NULL code object handle.

2.8.3 Enumeration Type Documentation

2.8.3.1 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 of the ELF shared object from which the code object was loaded. Note that the code object is the in memory loaded relocated form of the ELF shared object. Multiple code objects may be loaded at different memory addresses in the same process from the same ELF shared object.

The type of this attribute is a NUL terminated `char*`. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.
The URI name syntax is defined by the following BNF syntax:

```
code_object_uri ::== file_uri | memory_uri
file_uri        ::== "file://" file_path [ range_specifier ]
memory_uri      ::== "memory://" process_id range_specifier
range_specifier ::== [ "#" | "?" ] "offset=" number "&" "size=" number
file_path       ::== URI_ENCODED_OS_FILE_PATH
process_id      ::== DECIMAL_NUMBER
number          ::== HEX_NUMBER | DECIMAL_NUMBER | OCTAL_NUMBER
```

`number` is a C integral literal where hexadecimal values are prefixed by "0x" or "0X", and octal values by "0".

`file_path` is the file's path specified as a URI encoded UTF-8 string. In URI encoding, every character that is not in the regular expression `[a-zA-Z0-9/_.~-]` is encoded as two uppercase hexidecimal digits proceeded by "%". Directories in the path are separated by "/".

`offset` is a 0-based byte offset to the start of the code object. For a file URI, it is from the start of the file specified by the `file_path`, and if omitted defaults to 0. For a memory URI, it is the memory address and is required.

`size` is the number of bytes in the code object. For a file URI, if omitted it defaults to the size of the file. It is required for a memory URI.

`process_id` is the identity of the process owning the memory. For Linux it is the C unsigned integral decimal literal for the process ID (PID).
For example:

```
file:///dir1/dir2/file1
file:///dir3/dir4/file2#offset=0x2000&size=3000
memory://1234#offset=0x20000&size=3000
```

AMD_DBGAPI_CODE_OBJECT_INFO_LOAD_ADDRESS The difference between the address in the ELF shared object and the address the code object is loaded in memory. The type of this attributes is `ptrdiff_t`.

2.8.4 Function Documentation

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

Return values

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

2.8.4.2 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 amd_dbgapi_code_object_list then return NULL. Otherwise, return a pointer to an array of amd_dbgapi_code_object_id_t with `code_object_count` elements. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.
`in,out`	*changed*	If NULL then left unaltered. If non-NULL, set to AMD_DBGAPI_CHANGED_NO if the list of code objects is the same as when amd_dbgapi_code_object_list was last called, otherwise set to AMD_DBGAPI_CHANGED_YES.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `changed`, `code_object_count`, and `code_objects`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `code_object_count`, `code_objects`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `code_object_count`, `code_objects`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. `code_object_count`, `code_objects`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`code_object_count` or `code_objects` are NULL, or `changed` is invalid. `code_object_count`, `code_objects`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `code_objects` returns NULL. `code_object_count`, `code_objects`, and `changed` are unaltered.

2.9 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_DBG↩
  API_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_C↩
  OUNT = 6, AMD_DBGAPI_AGENT_INFO_COMPUTE_UNIT_COUNT = 7, AMD_DBGAPI_AGENT_INFO_N↩
  UM_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) AMD_↩
  DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VER↩
  SION_0_1

  Return the list of agents for a process.

2.9.1 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.

2.9.2 Macro Definition Documentation

2.9.2.1 #define AMD_DBGAPI_AGENT_NONE (amd_dbgapi_agent_id_t{ 0 })

The NULL agent handle.

2.9.3 Enumeration Type Documentation

2.9.3.1 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 poiter 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 (BDF) Notation][bfd]. 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`.

2.9.4 Function Documentation

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

Return values

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

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

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `changed`, `agent_count`, and `agents`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `agent_count`, `agents`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `agent_count`, `agents`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. `agent_count`, `agents`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`agent_count` or `agents` are NULL, or `changed` is invalid. `agent_count`, `agents`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `agents` returns NULL. `agent_count`, `agents`, and `changed` are unaltered.

2.10 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_HSA_KERNEL_DISPATCH_COOPERATIVE = 3, AMD_DBGAPI_QUEUE_TYPE_AMD_PM4 = 257 }

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Return the packets for a queue of a process.

2.10.1 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.

2.10.2 Macro Definition Documentation

2.10.2.1 #define AMD_DBGAPI_QUEUE_NONE (amd_dbgapi_queue_id_t{ 0 })

The NULL queue handle.

2.10.3 Typedef Documentation

2.10.3.1 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.

2.10.4 Enumeration Type Documentation

2.10.4.1 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.

2.10.4.2 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.

2.10.4.3 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.

2.10.4.4 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 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_HSA_KERNEL_DISPATCH_COOPERATIVE Queue supports HSA kernel dispatch with multiple producers protocol that supports cooperative dispatches. Queues of this type follow the same protocol as AMD_DBGAPI_QUEUE_TYPE_HSA_KERNEL_DISPATCH_MULTIPLE_PRODUCER. In addition, dispatches are able to use global wave synchronization (GWS) operations.

AMD_DBGAPI_QUEUE_TYPE_AMD_PM4 Queue supports the AMD PM4 protocol.

2.10.5 Function Documentation

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

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `value`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_QUEUE_ID	`queue_id` is invalid. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`value` is NULL or `query` is invalid. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE	`value_size` does not match the size of the result. `value` is unaltered.

Return values

AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `value` returns NULL. `value` is unaltered.

2.10.5.2 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`	*process_id*	The process for which the queue list is requested.
`out`	*queue_count*	The number of queues accessed by the process.
`out`	*queues*	If `changed` is not NULL and the queue list has not changed since the last call to amd_dbgapi_queue_list then return NULL. Otherwise, return a pointer to an array of amd_dbgapi_queue_id_t with `queue_count` elements. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.
`in,out`	*changed*	If NULL then left unaltered. If non-NULL, set to AMD_DBGAPI_CHANGED_NO if the list of queues is the same as when amd_dbgapi_queue_list was last called, otherwise set to AMD_DBGAPI_CHANGED_YES.

Return values

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

2.10.5.3 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	*process_id*	The process of the queue for which the packet list is requested.
in	*queue_id*	The queue for which the packet list is requested.
out	*first_packet_id*	The packet ID for the first packet in `packets_bytes`. If `packets_byte_size` is zero, then the packet ID for the next packet added to the queue.
out	*packets_byte_size*	The number of bytes of packets on the queue.
out	*packets_bytes*	A pointer to an array of `packets_byte_size` bytes. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `packets_byte_size` and `packets_bytes`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `packets_byte_size` and `packets_bytes` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `packets_byte_size` and `packets_bytes` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. `packets_byte_size` and `packets_bytes` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`packets_byte_size` or `packets_bytes` are NULL. `packets_byte_size` and `packets_bytes` are unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `packets_bytes` returns NULL. `packets_byte_size` and `packets_bytes` are unaltered.

2.11 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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Return the list of dispatches for a process.

2.11.1 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.

2.11.2 Macro Definition Documentation

2.11.2.1 #define AMD_DBGAPI_DISPATCH_NONE (amd_dbgapi_dispatch_id_t{ 0 })

The NULL dispatch handle.

2.11.3 Enumeration Type Documentation

2.11.3.1 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.

2.11.3.2 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.

2.11.3.3 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.

2.11.4 Function Documentation

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

Return values

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

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

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `changed`, `dispatch_count`, and `dispatches`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `changed`, `dispatch_count`, and `dispatches` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `changed`, `dispatch_count`, and `dispatches` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. `dispatch_count`, `dispatches`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`dispatch_count` or `dispatches` are NULL, or `changed` is invalid. `dispatch_count`, `dispatches`, and `changed` are unaltered.

2.12 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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Resume execution of a stopped wave.

2.12.1 Detailed Description

Operations related to AMD GPU waves.

2.12.2 Macro Definition Documentation

2.12.2.1 #define AMD_DBGAPI_WAVE_NONE (amd_dbgapi_wave_id_t{ 0 })

The NULL wave handle.

2.12.3 Enumeration Type Documentation

2.12.3.1 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.

2.12.3.2 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.

If the dispatch associated with a wave is not available then AMD_DBGAPI_DISPATCH_NONE is returned. If a wave has no associated dispatch then the the AMD_DBGAPI_WAVE_INFO_WORK_GROUP_COORD query may return incorrect information. Note that a wave may not have an associated dispatch if attaching to a process with already existing waves.

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`.

2.12.3.3 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.

2.12.3.4 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. Use the AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_PC_ADJUST query to determine the address of the 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 an 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.

2.12.4 Function Documentation

2.12.4.1 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`	*process_id*	The process to which the queue belongs.
`in`	*wave_id*	The handle of the wave being queried.
`in`	*query*	The query being requested.
`in`	*value_size*	Size of the memory pointed to by `value`. Must be equal to the byte size of the query result.
`out`	*value*	Pointer to memory where the query result is stored.

Return values

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

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

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `changed`, `wave_count`, and `waves`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `changed`, `wave_count`, and `waves` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `wave_count`, `waves`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. `wave_count`, `waves`, and `unchanged` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`wave_count` or `waves` are NULL, or `changed` is invalid. `wave_count`, `waves`, and `changed` are unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `waves` returns NULL. `wave_count`, `waves`, and `changed` are unaltered.

2.12.4.3 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_WAVE_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	*process_id*	The process to which the wave belongs.
in	*wave_id*	The wave being requested to resume.
in	*resume_mode*	If AMD_DBGAPI_RESUME_MODE_NORMAL, then resume normal execution of the wave. If AMD_DBGAPI_RESUME_MODE_SINGLE_STEP, then resume the wave in single step mode.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the wave will either terminate or be stopped. In either case a AMD_DBGAPI_EVENT_KIND_WAVE_STOP event will be reported.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized.

Return values

AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and no wave is resumed.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. No wave is resumed.
AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID	`wave_id` is invalid. No wave is resumed.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`resume_mode` is invalid. No wave is resumed.
AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED	`wave_id` is not stopped. The wave remains running.
AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_RESUMABLE	`wave_id` is stopped with a reason that includes one that cannot be resumed.

2.12.4.4 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	*process_id*	The process to which the wave belongs.
in	*wave_id*	The wave being requested to stop.

Return values

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

2.13 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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Complete a displaced stepping buffer for a wave.

2.13.1 Detailed Description

Operations related to AMD GPU breakpoint displaced stepping.

The library supports displaced stepping buffers. These allow an instruction that is overwritten by a breakpoint instruction to be copied to a buffer and single stepped in that buffer. This avoids needing to remove the breakpoint instruction by replacing it with the original instruction bytes, single stepping the original instruction, and finally restoring the breakpoint instruction.

This allows a client to support non-stop debugging where waves are left executing while others are halted after hitting a breakpoint. If resuming from a breakpoint involved removing the breakpoint, it could result in the running waves missing the removed breakpoint.

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 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 unspecified, 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. In general, it is best for the client to single step as many waves as possible to minimize the time to get all waves stepped over the breakpoints.

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, or through buffers being associated with specific agents or queues.

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 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.

If allocating a displaced stepping buffer indicates no more are available, the client must complete using the previously allocated buffers. It can do that by ensuring the allocated waves are resumed in single step mode, ensure that the waves will make forward progress, and process any reported pending events. This allows waves to perform the single step, report the single step has completed by an event, and the client's processing of the event will complete the displaced stepping buffer. That may free up a displaced stepping buffer for use by the client for other waves. Since there is always at least one displaced stepping buffer, in general, the worst case is that one wave at a time can be single stepped over a breakpoint using a displaced stepping buffer.

However, the weak forward progress of AMD GPU execution can result in no waves that have successfully been allocated a displaced stepping buffer from actually reporting completion of the single step. For example, this can happen if the waves being single stepped are prevented from becoming resident on the hardware due to other waves that are halted. The waves being single stepped can be stopped before completing the single step to release the displaced stepping buffer for use by a different set of waves. In the worst case, the user may have to continue halted waves and allow them to terminate before other waves can make forward progress to complete the single step using a displaced stepping buffer.

See also

amd_dbgapi_wave_resume, amd_dbgapi_wave_stop, amd_dbgapi_process_set_progress, amd_dbgapi_next_↩ pending_event

2.13.2 Macro Definition Documentation

2.13.2.1 #define AMD_DBGAPI_DISPLACED_STEPPING_NONE (amd_dbgapi_displaced_stepping_id_t{ 0 })

The NULL displaced stepping handle.

2.13.3 Function Documentation

2.13.3.1 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	*process_id*	The process to which the wave belongs.
in	*wave_id*	The wave using the displaced stepping buffer.
in	*displaced_stepping*	The displaced stepping buffer to complete.

Return values

AMD_DBGAPI_STATUS_SUCCESS	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.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized, and no displaced stepping buffer is completed.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized, no displaced stepping buffer completed.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. No displaced stepping buffer is completed.
AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID	`wave_id` is invalid. No displaced stepping buffer is completed.
AMD_DBGAPI_STATUS_ERROR_INVALID_DISPLACED_STEPPING_ID	`displaced_stepping` is invalid or not in use by `wave_id`. No displaced stepping buffer is completed.
AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED	`wave_id` is not stopped. No displaced stepping buffer is completed.

2.13.3.2 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	*process_id*	The process to which the wave belongs.
in	*wave_id*	The wave to create a displaced stepping buffer.
in	*saved_instruction_bytes*	The original instruction bytes that the breakpoint instruction replaced. The number of bytes must be AMD_DBGAPI_ARCHITECTURE_INFO_BREAKPOINT_INSTRUCTION_SIZE.
out	*displaced_stepping*	The displace stepping handle, or AMD_DBGAPI_DISPLACED_STEPPING_NONE.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and `displaced_stepping` is set to AMD_DBGAPI_DISPLACED_STEPPING_NONE or to a valid displaced stepping handle.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized, no displaced stepping buffer is allocated, and `displaced_stepping` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized, no displaced stepping buffer is allocated, and `displaced_stepping` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. No displaced stepping buffer is allocated and `displaced_stepping` is unaltered.

Return values

AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID	`wave_id` is invalid. No displaced stepping buffer is allocated and `displaced_stepping` is unaltered.
AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED	`wave_id` is not stopped. No displaced stepping buffer is allocated and `displaced_stepping` is unaltered.
AMD_DBGAPI_STATUS_ERROR_DISPLACED_STEPPING_BUFFER_UNAVAILABLE	No more displaced stepping buffers are available that are suitable for use by `wave_id`. No displaced stepping buffer is allocated and `displaced_stepping` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`original_instruction` or `displaced_stepping` are NULL. No displaced stepping buffer is allocated and `displaced_stepping` is unaltered.

2.14 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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Remove a hardware data watchpoint previously set by amd_dbgapi_set_watchpoint.

2.14.1 Detailed Description

Operations related to AMD GPU hardware data watchpoints.

A data watchpoint is a hardware supported mechanism to generate wave stop events after a wave accesses memory in a certain way in a certain address range. The memory access will have been completed before the event is reported.

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.

2.14.2 Macro Definition Documentation

2.14.2.1 #define AMD_DBGAPI_WATCHPOINT_NONE ((amd_dbgapi_watchpoint_id_t) (-1))

The NULL watchpoint handle.

2.14.3 Typedef Documentation

2.14.3.1 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.

2.14.4 Enumeration Type Documentation

2.14.4.1 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.

2.14.4.2 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 The watchpoints of an agent are not shared across processes. Every process using an agent can use all AMD_DBGAPI_ARCHITECTURE_INFO_W↩ ATCHPOINT_COUNT watchpoints.

AMD_DBGAPI_WATCHPOINT_SHARE_KIND_SHARED The watchpoints of an agent are shared between all processes using the agent. The number of watchpoints for an agent available to a process may be reduced if watchpoints for that agent are used by another process.

2.14.5 Function Documentation

2.14.5.1 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	*process_id*	The process to which the agent belongs.
in	*agent_id*	Specify the agent that owns the watchpoint.
in	*watchpoint_id*	The watchpoint to remove.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the watchpoint has been removed.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and no watchpoint is removed.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. No watchpoint is removed.
AMD_DBGAPI_STATUS_ERROR_INVALID_AGENT_ID	`agent_id` is invalid. No watchpoint is removed.
AMD_DBGAPI_STATUS_ERROR_INVALID_WATCHPOINT_ID	`watchpoint_id` is invalid. No watchpoint is removed.

2.14.5.2 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	*process_id*	The process to which the agent belongs.
in	*agent_id*	Specify the agent to set the watchpoint.
in	*address*	The base address of memory area to set a watchpoint.
in	*size*	The number of bytes that the watchpoint should cover.
in	*kind*	The kind of memory access that should trigger the watchpoint.
out	*watchpoint_id*	The watchpoint created.
out	*watchpoint_address*	The base address of the created watchpoint.
out	*watchpoint_size*	The byte size of the created watchpoint.

Return values

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

2.15 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_REGISTE↩ R_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) AMD_DBGAPI_VERSION_0_1

  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) AMD↩ _DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Prefetch register values.

2.15.1 Detailed Description

Operations related to AMD GPU register access.

2.15.2 Macro Definition Documentation

2.15.2.1 #define AMD_DBGAPI_REGISTER_CLASS_NONE (amd_dbgapi_register_class_id_t{ 0 })

The NULL register class handle.

2.15.2.2 #define AMD_DBGAPI_REGISTER_NONE (amd_dbgapi_register_id_t{ 0 })

The NULL register handle.

2.15.3 Enumeration Type Documentation

2.15.3.1 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.

2.15.3.2 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.

2.15.3.3 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. `unit64` 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.

2.15.4 Function Documentation

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

Return values

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

2.15.4.2 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	*architecture_id*	The architecture being queried.
out	*register_class_count*	The number of architecture register classes.
out	*register_classes*	A pointer to an array of amd_dbgapi_register_class_id_t with register_class_count elements. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.

Return values

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

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

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `value`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`wave_id` is invalid. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID	`register_id` is invalid for `architecture_id`. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`value` is NULL, or `query` is invalid or not supported for an architecture. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE	`value_size` does not match the size of the result. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `value` returns NULL. `value` is unaltered.

2.15.4.4 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`	*architecture_id*	The architecture being queried.
`out`	*register_count*	The number of architecture registers.
`out`	*registers*	A pointer to an array of amd_dbgapi_register_id_t with `register_count` elements. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `register_count` and `registers`.

Return values

AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `register_count` and `registers` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `register_count` and `registers` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `register_count` and `registers` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`register_count` or `registers` are NULL. `register_count` and `registers` are unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `registers` returns NULL. `register_count` and `registers` are unaltered.

2.15.4.5 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`	*architecture_id*	The architecture of the DWARF register.
`in`	*dwarf_register*	The AMD GPU DWARF register number.
`out`	*register_id*	The register handle that corresponds to the DWARF register ID.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `register_id`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `register_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `register_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `register_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`dwarf_register_id` is not valid for the architecture or `register_id` is NULL. `register_id` is unaltered.

2.15.4.6 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	*process_id*	The process to which the wave belongs.
in	*wave_id*	The wave being queried for the register.
in	*register_id*	The first register being requested.
in	*register_count*	The number of registers being requested.

Return values

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

2.15.4.7 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`	*process_id*	The process to which the wave belongs.
`in`	*wave_id*	The wave to being queried for the register.
`in`	*register_id*	The register being requested.
`in`	*offset*	The first byte to start reading the register. The offset is zero based starting from the least significant byte of the register.
`in`	*value_size*	The number of bytes to read from the register which must be greater than 0 and less than the size of the register minus `offset`.
`out`	*value*	The bytes read from the register. Must point to an array of at least `value_size` bytes.

Return values

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

2.15.4.8 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*	AMD_DBGAPI_REGISTER_CLASS_STATE_NOT_MEMBER if the register is not in the register class. AMD_DBGAPI_REGISTER_CLASS_STATE_MEMBER if the register is in the register class.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `register_class_state`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `register_class_state` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `register_class_state` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `register_class_state` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_ID	`register_id` is invalid for `architecture_id`. `register_class_state` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_REGISTER_CLASS_ID	`register_class_id` is invalid for `architecture_id`. `register_class_state` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`register_class_state` is NULL. `register_class_state` is unaltered.

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

Return values

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

2.15.4.10 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 amd_dbgapi_register_id_t with `register_count` elements. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `register_count` and `registers`.

Return values

AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `register_count` and `registers` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `register_count` and `registers` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `register_count` and `registers` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`register_count` or `registers` are NULL. `register_count` and `registers` are unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `registers` returns NULL. `register_count` and `registers` are unaltered.

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

Return values

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

2.16 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_DBG↩
  API_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_DB↩
  GAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE = 2, AMD_DBGAPI_ADDRESS_SPACE_INFO_NULL_A↩
  DDRESS = 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_D↩
  BGAPI_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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

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) AMD_DBGAPI_VERSION_0_1

  Control precision of memory access reporting.

2.16.1 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.

2.16.2 Macro Definition Documentation

2.16.2.1 #define AMD_DBGAPI_ADDRESS_CLASS_NONE (amd_dbgapi_address_class_id_t{ 0 })

The NULL address class handle.

2.16.2.2 #define AMD_DBGAPI_LANE_NONE ((amd_dbgapi_lane_id_t) (-1))

The NULL lane handle.

2.16.3 Typedef Documentation

2.16.3.1 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 generated 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.

2.16.3.2 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 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.

2.16.4 Enumeration Type Documentation

2.16.4.1 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.

2.16.4.2 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.

2.16.4.3 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 lifetie 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.

2.16.4.4 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.

2.16.4.5 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.

2.16.4.6 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.

2.16.5 Function Documentation

2.16.5.1 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	*process_id*	The process to which the `wave_id` belongs.
in	*wave_id*	The wave that is using the address.
in	*lane_id*	The lane of the `wave_id` that is using the address.
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 AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE query.
in	*address_class_id*	The handle of the source language address class.
out	*address_class_state*	AMD_DBGAPI_ADDRESS_CLASS_STATE_NOT_MEMBER if the address is not in the address class. AMD_DBGAPI_ADDRESS_CLASS_STATE_MEMBER if the address is in the address class.

Return values

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

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

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `value`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `value` is unaltered.

Return values

AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID	`address_space_id` is invalid for `architecture_id`. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`query` is invalid or `value` is NULL. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE	`value_size` does not match the size of the result. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `value` returns NULL. `value` is unaltered.

2.16.5.3 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.

If addresses in one address space may alias the addresses in another, and if memory locations are updated using an address in one, then any cached information about values in the other needs to be invalidated.

The address spaces must match the architecture.

Parameters

`in`	*architecture_id*	The architecture to which the address spaces belong.
`in`	*address_space_id1*	An address space.
`in`	*address_space_id2*	An address space.
`out`	*address_space_alias*	AMD_DBGAPI_ADDRESS_SPACE_ALIAS_NONE if the address spaces do not alias. AMD_DBGAPI_ADDRESS_SPACE_ALIAS_MAY if the address spaces may alias.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `address_space_alias`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `address_space_alias` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `address_space_alias` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `address_space_alias` is unaltered.

Return values

AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID	`address_space_id1` or `address_space_id2` are invalid for `architecture_id`. `address_space_alias` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`address_space_alias` is NULL. `address_space_alias` is unaltered.

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

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `value`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_CLASS_ID	`address_class_id` is invalid for the architecture of `architecture_id`. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`value` is NULL or `query` is invalid. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT_SIZE	`value_size` does not match the size of the result. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `value` returns NULL. `value` is unaltered.

2.16.5.5 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	*architecture_id*	The architecture being queried.
out	*address_class_count*	The number of architecture source language address classes.
out	*address_classes*	A pointer to an array of amd_dbgapi_address_class_id_t with address_class_count elements. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in address_class_count and address_classes.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and address_class_count and address_classes are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and address_class_count and address_classes are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHI↩ TECTURE_ID	architecture_id is invalid. address_class_count and address_classes are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGU↩ MENT	address_class_count or address_classes are NULL. address_class_count and address_classes are unaltered.
AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBA↩ CK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate address_classes returns NULL. address_class_count and address_classes are unaltered.

2.16.5.6 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	*architecture_id*	The architecture being queried.
out	*address_space_count*	The number of architecture address spaces.
out	*address_spaces*	A pointer to an array of amd_dbgapi_address_space_id_t with address_space_count elements. It is allocated by the amd_dbgapi_callbacks_s::allocate_memory callback and is owned by the client.

Return values

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

2.16.5.7 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	*process_id*	The process to which the wave_id belongs.
in	*wave_id*	The wave that is using the address.
in	*lane_id*	The lane of the wave_id that is using the address.
in	*source_address_space*	The address space of the source_segment_address.
in	*source_segment_address*	The integral value of the source segment address. Only the bits corresponding to the address size for the source_address_space requested are used. The address size is provided by the AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE query.
in	*destination_address_space*	The address space to which to convert source_segment_address that is in source_address_space.
out	*destination_segment_address*	The integral value of the segment address in destination_address_space that corresponds to source_segment_address in source_address_space. The bits corresponding to the address size for the destination_address_space are updated, and any remaining bits are set to zero. The address size is provided by the AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE query.

Return values

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

2.16.5.8 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	*architecture_id*	The architecture of the source language address class.
in	*dwarf_address_class*	The DWARF source language address class.
out	*address_class_id*	The source language address class that corresponds to the DWARF address class for the architecture.

Return values

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

2.16.5.9 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	*architecture_id*	The architecture of the address space.
in	*dwarf_address_space*	The AMD GPU DWARF address space.
out	*address_space_id*	The address space that corresponds to the DWARF address space for the architecture architecture_id.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the result is stored in `address_space_id`.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `address_space_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `address_space_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARCHITECTURE_ID	`architecture_id` is invalid. `address_space_id` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`dwarf_address_space` is not valid for `architecture_id`, or `address_space_id` is NULL. `address_space_id` is unaltered.

2.16.5.10 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`	*process_id*	The process to which the `wave_id` belongs.
`in`	*wave_id*	The wave that is reading 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 AMD_DBGAPI_LANE_NONE.
`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 AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE query.
`in,out`	*value_size*	Pass in the number of bytes to read from memory. Return the number of bytes successfully read from memory.
`out`	*value*	Pointer to memory where the result is stored. Must be an array of at least input `value_size` bytes.

Return values

AMD_DBGAPI_STATUS_SUCCESS	Either the input `value_size` was 0, or the input `value_size` was greater than 0 and one or more bytes have been read successfully. The output `value_size` is set to the number of bytes successfully read, which will be 0 if the input `value_size` was 0. The first output `value_size` bytes of `value` are set to the bytes successfully read, all other bytes in `value` are unaltered.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `value_size` and `value` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `value_size` and `value` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. `value_size` and `value` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_WAVE_ID	`wave_id` is invalid. `value_size` and `value` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_LANE_ID	`lane_id` is invalid, or `lane_id` is AMD_DBGAPI_LANE_NONE and `address_space` depends on the active lane. `value_size` and `value` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ADDRESS_SPACE_ID	`address_space_id` is invalid for the architecture of `wave_id`. `value` is unaltered.
AMD_DBGAPI_STATUS_ERROR_WAVE_NOT_STOPPED	`wave_id` is not stopped. `value_size` and `value` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`value` or `value_size` are NULL. `value_size` and `value` are unaltered.
AMD_DBGAPI_STATUS_ERROR_MEMORY_ACCESS	The input `value_size` was greater than 0 and no bytes were successfully read. The output `value_size` is set to 0. All bytes in `value` are unaltered.

2.16.5.11 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. The setting may be changed at any time, including when waves are executing, and takes effect immediately.

Parameters

`in`	*process_id*	The process being configured.
`in`	*agent_id*	The agent to configure.
`in`	*memory_precision*	The memory precision to set.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the agent has been configured.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and no agent configuration is changed.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	`process_id` is invalid. No agent configuration is changed.
AMD_DBGAPI_STATUS_ERROR_INVALID_AGENT_ID	`agent_id` is invalid. No agent configuration is changed.
AMD_DBGAPI_STATUS_ERROR_NOT_SUPPORTED	The requested `memory_precision` is not supported for the architecture of the agent. No agent configuration is changed.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`memory_precision` is an invalid value. No agent configuration is changed.

2.16.5.12 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 writting 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 AMD_DBGAPI_LANE_NONE.
`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 AMD_DBGAPI_ADDRESS_SPACE_INFO_ADDRESS_SIZE 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

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

2.17 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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Report that an event has been processed.

2.17.1 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

2.17.2 Macro Definition Documentation

2.17.2.1 #define AMD_DBGAPI_EVENT_NONE (amd_dbgapi_event_id_t{ 0 })

The NULL event handle.

2.17.3 Enumeration Type Documentation

2.17.3.1 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.

2.17.3.2 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.

2.17.3.3 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.

2.17.4 Function Documentation

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

Return values

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

Return values

AMD_DBGAPI_STATUS_ERROR_CLIENT_CALLBACK	This will be reported if the amd_dbgapi_callbacks_s::allocate_memory callback used to allocate `value` returns NULL. `value` is unaltered.

2.17.4.2 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`	*process_id*	The process to which `event_id` belongs.
`in`	*event_id*	The event that has been processed.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and the event has been reported as processed. The `event_id` is invalidated.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	The `process_id` is invalid. No event is marked as processed.
AMD_DBGAPI_STATUS_ERROR_INVALID_EVENT_ID	The `event_id` is invalid or the NULL event. No event is marked as processed.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`event_id` or `kind` are NULL. No event is marked as processed.

2.17.4.3 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`	*process_id*	The process from which to retrieve pending events.
`out`	*event_id*	The event handle of the next pending event. Each event is only returned once. If there are no pending events the AMD_DBGAPI_EVENT_NONE handle is returned.

Parameters

`out`	*kind*	The kind of the returned event. If there are no pending events, then AMD_DBGAPI_EVENT_KIND_NONE is returned.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and an event or the NULL event has been returned.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized; and `event_id` and `kind` are unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized; and `event_id` and `kind` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	The `process_id` is invalid. No event is retrieved and `event_id` and `kind` are unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`event_id` or `kind` are NULL. No event is retrieved and `event_id` and `kind` are unaltered.

2.18 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) AMD_DBGAPI_VERSION_0_1
  Set the logging level.

2.18.1 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.

2.18.2 Enumeration Type Documentation

2.18.2.1 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.

2.18.3 Function Documentation

2.18.3.1 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

`in`	*level*	The logging level to set.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`level` is invalid. The logging level is ot changed.

2.19 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 struct amd_dbgapi_client_thread_s * 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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Report that a breakpoint added by the amd_dbgapi_callbacks_s::add_breakpoint calback has been hit.

2.19.1 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.

2.19.2 Macro Definition Documentation

2.19.2.1 #define AMD_DBGAPI_BREAKPOINT_NONE ((amd_dbgapi_breakpoint_id_t) (0))

The NULL breakpoint handle.

2.19.2.2 #define AMD_DBGAPI_SHARED_LIBRARY_NONE (amd_dbgapi_shared_library_id_t{ 0 })

The NULL shared library handle.

2.19.3 Typedef Documentation

2.19.3.1 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.

2.19.3.2 typedef struct amd_dbgapi_client_thread_s* 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.

2.19.4 Enumeration Type Documentation

2.19.4.1 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.

2.19.4.2 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.

2.19.4.3 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.

2.19.5 Function Documentation

2.19.5.1 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 calback 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 :A↩ MD_DBGAPI_BREAKPOINT_ACTION_HALT then the client should process pending events which will cause a AM↩ D_DBGAPI_EVENT_KIND_BREAKPOINT_RESUME event to be added which specifies that the thread should now be resumed.

Parameters

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

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully and `breakpoint_action` indicates if the thread hitting the breakpoint should be resumed.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The library is left uninitialized and `breakpoint_action` is unaltered.
AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The library is not initialized. The library is left uninitialized and `breakpoint_action` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	The `process_id` is invalid. `breakpoint_action` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_BREAKPOINT_ID	The `breakpoint_id` is invalid. `breakpoint_action` is unaltered.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`breakpoint_action` is NULL. `breakpoint_action` is unaltered.

2.19.5.2 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	*process_id*	The process to which the `shared_library_id` belongs.
in	*shared_library_id*	The shared library that has been loaded or unloaded.
in	*shared_library_state*	The shared library state.

Return values

AMD_DBGAPI_STATUS_SUCCESS	The function has been executed successfully.
AMD_DBGAPI_STATUS_FATAL	A fatal error occurred. The amd-dbgapi library is left uninitialized and `resume` is unaltered.

Return values

AMD_DBGAPI_STATUS_ERROR_NOT_INITIALIZED	The amd-dbgapi library is not initialized. The amd-dbgapi library is left uninitialized.
AMD_DBGAPI_STATUS_ERROR_INVALID_PROCESS_ID	The `process_id` is invalid.
AMD_DBGAPI_STATUS_ERROR_INVALID_SHARED_LIBRARY_ID	The `shared_library_id` is invalid.
AMD_DBGAPI_STATUS_ERROR_INVALID_ARGUMENT	`shared_library_state` is invalid.
AMD_DBGAPI_STATUS_ERROR	`shared_library_state` 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.

Every breakpoint added within a process will have a unique 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. Each added breakpoint for a process will have a unique handle, multiple breakpoints for the same process will not be added with the same handle. It 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_id` handle is invalid. No breakpoint is added.

Return AMD_DBGAPI_STATUS_ERROR_LIBRARY_NOT_LOADED if `shared_library_id` 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 `byte_size` is 0 or if unable to allocate memory of the byte size specified by `byte_size` then return NULL and allocate no memory. 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` will be a pointer returned by amd_dbgapi_callbacks_s::allocate_memory that will not be returned to the client. If `data` is NULL then it indicates the allocation failed or was for 0 bytes: in either case the callback is required to take no action. If `data` is not NULL then it will not have been deallocated by a previous call to amd_dbgapi_callbacks_s::allocate_memory. Note that these requirements can be met by implementing using `free`.

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_id` 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` has invalid library name syntax.

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 ∗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® 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_id` handle is invalid.

Return AMD_DBGAPI_STATUS_ERROR_LIBRARY_NOT_LOADED if `shared_library_id` 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

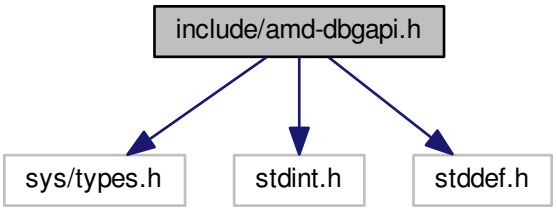
File Documentation

4.1 include/amd-dbgapi.h File Reference

AMD debugger API interface.

```
#include <sys/types.h>
#include <stdint.h>
#include <stddef.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_VERSION_0_1

  The function was introduced in version 0.1 of the interface and has the symbol version string of "AMD_DBGAPI_0.1".
- #define AMD_DBGAPI_VERSION_0_20

  The function was introduced in version 0.20 of the interface and has the symbol version string of "AMD_DBGAPI_0.20".
- #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 struct amd_dbgapi_symbolizer_id_s * amd_dbgapi_symbolizer_id_t

  Opaque client symbolizer handle.
- typedef struct amd_dbgapi_client_process_s * 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 struct amd_dbgapi_client_thread_s * 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_DBGAPI_STATUS_ERROR_UNIMPLEMENTED = INT32_MIN }

  AMD debugger API status codes.

- enum { AMD_DBGAPI_VERSION_MAJOR = 0, AMD_DBGAPI_VERSION_MINOR = 20 }

  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 = 6, 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.

- enum amd_dbgapi_instruction_kind_t { AMD_DBGAPI_INSTRUCTION_KIND_UNKNOWN = 0, AMD_DBGAPI_INSTRUCTION_KIND_SEQUENTIAL = 1, AMD_DBGAPI_INSTRUCTION_KIND_DIRECT_BRANCH = 2, AMD_DBGAPI_INSTRUCTION_KIND_DIRECT_BRANCH_CONDITIONAL = 3, AMD_DBGAPI_INSTRUCTION_KIND_INDIRECT_BRANCH_REGISTER_PAIR = 4, AMD_DBGAPI_INSTRUCTION_KIND_DIRECT_CALL_REGISTER_PAIR = 5, AMD_DBGAPI_INSTRUCTION_KIND_INDIRECT_CALL_REGISTER_PAIRS = 6, AMD_DBGAPI_INSTRUCTION_KIND_TERMINATE = 7, AMD_DBGAPI_INSTRUCTION_KIND_TRAP = 8, AMD_DBGAPI_INSTRUCTION_KIND_HALT = 9, AMD_DBGAPI_INSTRUCTION_KIND_BARRIER = 10, AMD_DBGAPI_INSTRUCTION_KIND_SLEEP = 11, AMD_DBGAPI_INSTRUCTION_KIND_SPECIAL = 12 }

  The kinds of instruction classifications.
- 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_wave_creation_t { AMD_DBGAPI_WAVE_CREATION_NORMAL = 0, AMD_DBGAPI_WAVE_CREATION_STOP = 1 }

  The kinds of wave creation supported by the hardware.
- 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_HSA_KERNEL_DISPATCH_COOPERATIVE = 3, AMD_DBGAPI_QUEUE_TYPE_AMD_PM4 = 257 }

  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) AMD_DBGAPI_VERSION_0_1

  Query a textual description of a status code.
- void AMD_DBGAPI amd_dbgapi_get_version (uint32_t *major, uint32_t *minor, uint32_t *patch) AMD_DBGAPI_VERSION_0_1

  Query the version of the installed library.
- const char AMD_DBGAPI * amd_dbgapi_get_build_name (void) AMD_DBGAPI_VERSION_0_1

  Query the installed library build name.
- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_initialize (amd_dbgapi_callbacks_t *callbacks) AMD_DBGAPI_VERSION_0_1

  Initialize the library.
- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_finalize (void) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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, amd_dbgapi_symbolizer_id_t symbolizer_id, amd_dbgapi_status_t(*symbolizer)(amd_dbgapi_symbolizer_id_t symbolizer_id, amd_dbgapi_global_address_t address, char **symbol_text)) AMD_DBGAPI_VERSION_0_20

  Disassemble a single instruction.

- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_classify_instruction (amd_dbgapi_architecture_id_t architecture_id, amd_dbgapi_global_address_t address, amd_dbgapi_size_t *size, const void *memory, amd_dbgapi_instruction_kind_t *instruction_kind, void **instruction_properties) AMD_DBGAPI_VERSION_0_20

  Classify 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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Detach from a process and no longer have debug control of the AMD GPU devices it uses.
- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_set_progress (amd_dbgapi_process_id_t process_id, amd_dbgapi_progress_t progress) AMD_DBGAPI_VERSION_0_1

  Set the progress required for a process.
- amd_dbgapi_status_t AMD_DBGAPI amd_dbgapi_process_set_wave_creation (amd_dbgapi_process_id_t process_id, amd_dbgapi_wave_creation_t creation) AMD_DBGAPI_VERSION_0_20

  Set the wave creation mode 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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  Report that an event has been processed.

- void AMD_DBGAPI amd_dbgapi_set_log_level (amd_dbgapi_log_level_t level) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

  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) AMD_DBGAPI_VERSION_0_1

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