ROCR Documentation

Release 1.15.0

Advanced Micro Devices, Inc.

Jul 22, 2025

INSTALL

The ROCm runtime (ROCR) is AMD's implementation of HSA runtime, which is a thin, user-mode API that exposes the necessary interfaces to access and interact with graphics hardware driven by the AMDGPU driver set and the ROCK kernel driver. To learn more, see what-is-rocr-runtime

You can access ROCR code on our GitHub repository.

The documentation is structured as follows:

Install

- *Installation*

API reference

- *C interface adaptors*
- *Environment variables*
- *API*

Contribution

- *Contributing to ROCR*

To contribute to the documentation, refer to Contributing to ROCm.

You can find licensing information on the Licensing page.

CHAPTER

ONE

INSTALLATION

This document provides information required to build and install ROCR using prebuilt binaries or from source.

1.1 Build and install using prebuilt binaries

Here is how you can install ROCR using prebuilt binaries.

1.1.1 Prerequisites

- A system supporting ROCm. See the supported operating systems.
- Install ROCm. See how to install ROCm.
- Install `libdrm` package.

```
sudo apt install libdrm-dev
```

The ROCR prebuilt binaries include:

Core runtime package:

- HSA include files to support application development on the HSA runtime for the ROCR runtime
- A 64-bit version of AMD's HSA core runtime for the ROCR runtime

Runtime extension package:

- A 64-bit version of AMD's runtime tools library
- A 64-bit version of AMD's runtime image library

The contents of these packages are installed in `/opt/rocm/hsa` and `/opt/rocm` by default. The core runtime package depends on the `hsakmt-roct-dev` package.

1.2 Build and install from source

Here is how you can build ROCR from source.

1.2.1 Prerequisites

- CMake 3.7 or later. Export CMake bin into your PATH.
- Support packages `libelf-dev` and `g++`.

```
sudo apt install libelf-dev g++
```

- A compatible version of the `libhsakmt` library and the `hsakmt.h` header file. Obtain the latest version of these files from the ROCT-Thunk-Interface repository.
- Install `xxd`.

```
sudo apt install xxd
```

1.3 Building the runtime

The `libhsakmt` development packages include a CMake package config file. The runtime locates `libhsakmt` via `find_package` if `libhsakmt` is installed in a standard location. For installations that don't use standard ROCm paths, set CMake variables `CMAKE_PREFIX_PATH` or `hsakmt_DIR` to override `find_package` search paths. The runtime includes an optional image support module (previously `hsa-ext-rocr-dev`). By default this module is included in the runtime builds. To exclude the image module from the runtime, set the CMake variable `IMAGE_SUPPORT` to OFF. To build the optional image module, install AMDGCN-compatible clang and device library. You can find the latest version of these additional build dependencies in the ROCm package repository. The latest source for these projects are available in the llvm project and ROCm device libs repositories.

The runtime optionally supports use of the CMake user package registry. By default the registry is not modified. Set CMake variable `EXPORT_TO_USER_PACKAGE_REGISTRY` to ON to enable updating the package registry.

To build, install, and produce packages on a system with standard ROCm packages installed, clone your copy of ROCR and run the following from `src/`:

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/opt/rocm ..
make
make install
make package
```

Example with a custom installation path, build dependency path, and options:

```
cmake -DIMAGE_SUPPORT=OFF \
      -DEXPORT_TO_USER_PACKAGE_REGISTRY=ON \
      -DCMAKE_VERBOSE_MAKEFILE=1 \
      -DCMAKE_PREFIX_PATH=<alternate path(s) to build dependencies> \
      -DCMAKE_INSTALL_PATH=<custom install path for this build> \
      ..
```

Alternatively, use `ccmake` and `cmake-gui`:

```
mkdir build
cd build
ccmake ..
press c to configure
populate variables as desired
press c again
press g to generate and exit
make
```

1.4 Building against the runtime

The runtime provides a CMake package config file, installed by default to `/opt/rocm/lib/cmake/hsa-runtime64`. The runtime exports CMake target `hsa-runtime64` in namespace `hsa-runtime64`. A CMake project (`Foo`) using the runtime may locate, include, and link the runtime using the following template:

```
# Add /opt/rocm to CMAKE_PREFIX_PATH.

find_package(hsa-runtime64 1.0 REQUIRED)
...
add_library(Foo ...)
...
target_link_libraries(Foo PRIVATE hsa-runtime64::hsa-runtime64)
```

CHAPTER

TWO

C INTERFACE ADAPTORS

The C interface layer is the *top layer in ROCR* that provides C++ APIs as defined in the HSA Runtime Specification 1.2. The C interface layer also consists of the interfaces and default definitions for the standard extensions. The interface functions simply forward to a function pointer table defined here. The table is initialized to point to default definitions, which simply returns an appropriate error code. If available, the extension library is loaded as part of runtime initialization and the table is updated to point to the extension library.

Files present in this layer:

- `hsa.h` (cpp)
- `hsa_ext_interface.h` (cpp)

CHAPTER

THREE

ENVIRONMENT VARIABLES

The following table lists the most often used environment variables.

Table 3.1: ROCR environment variables

Environment variable	Possible values	Description
HSA_ENABLE_SDMA	• 0: Disabled • 1: Enabled (default)	This controls the use of DMA engines in all copy directions (Host-to-Device, Device-to-Host, Device-to-Device) when using the `hsa_memory_copy`, `hsa_amd_memory_fill`, `hsa_amd_memory_async_copy`, `hsa_amd_memory_async_copy_on_engine` APIs
HSA_ENABLE_PEER_SDMA	• 0: Disabled • 1: Enabled (default)	This controls the use of DMA engines for Device-to-Device copies when using the `hsa_memory_copy`, `hsa_amd_memory_async_copy`, `hsa_amd_memory_async_copy_on_engine` APIs

Note

The value of `HSA_ENABLE_PEER_SDMA` is ignored if `HSA_ENABLE_SDMA` is used to disable the use of DMA engines.

CHAPTER

FOUR

API

genindex

search

4.1 Architected Queuing Language

enum **hsa_packet_type_t**

Packet type.

Values:

enumerator **HSA_PACKET_TYPE_VENDOR_SPECIFIC**

Vendor-specific packet.

enumerator **HSA_PACKET_TYPE_INVALID**

The packet has been processed in the past, but has not been reassigned to the packet processor. A packet processor must not process a packet of this type. All queues support this packet type.

enumerator **HSA_PACKET_TYPE_KERNEL_DISPATCH**

Packet used by agents for dispatching jobs to kernel agents. Not all queues support packets of this type (see *hsa_queue_feature_t*).

enumerator **HSA_PACKET_TYPE_BARRIER_AND**

Packet used by agents to delay processing of subsequent packets, and to express complex dependencies between multiple packets. All queues support this packet type.

enumerator **HSA_PACKET_TYPE_AGENT_DISPATCH**

Packet used by agents for dispatching jobs to agents. Not all queues support packets of this type (see *hsa_queue_feature_t*).

enumerator **HSA_PACKET_TYPE_BARRIER_OR**

Packet used by agents to delay processing of subsequent packets, and to express complex dependencies between multiple packets. All queues support this packet type.

enum **hsa_fence_scope_t**

Scope of the memory fence operation associated with a packet.

Values:

enumerator **HSA_FENCE_SCOPE_NONE**

No scope (no fence is applied). The packet relies on external fences to ensure visibility of memory updates.

enumerator **HSA_FENCE_SCOPE_AGENT**

The fence is applied with agent scope for the global segment.

enumerator **HSA_FENCE_SCOPE_SYSTEM**

The fence is applied across both agent and system scope for the global segment.

enum **hsa_packet_header_t**

Sub-fields of the *header* field that is present in any AQL packet. The offset (with respect to the address of *header*) of a sub-field is identical to its enumeration constant. The width of each sub-field is determined by the corresponding value in *hsa_packet_header_width_t*. The offset and the width are expressed in bits.

Values:

enumerator **HSA_PACKET_HEADER_TYPE**

Packet type. The value of this sub-field must be one of *hsa_packet_type_t*. If the type is *HSA_PACKET_TYPE_VENDOR_SPECIFIC*, the packet layout is vendor-specific.

enumerator **HSA_PACKET_HEADER_BARRIER**

Barrier bit. If the barrier bit is set, the processing of the current packet only launches when all preceding packets (within the same queue) are complete.

enumerator **HSA_PACKET_HEADER_SCACQUIRE_FENCE_SCOPE**

Acquire fence scope. The value of this sub-field determines the scope and type of the memory fence operation applied before the packet enters the active phase. An acquire fence ensures that any subsequent global segment or image loads by any unit of execution that belongs to a dispatch that has not yet entered the active phase on any queue of the same kernel agent, sees any data previously released at the scopes specified by the acquire fence. The value of this sub-field must be one of *hsa_fence_scope_t*.

enumerator **HSA_PACKET_HEADER_ACQUIRE_FENCE_SCOPE**

Deprecated:

Renamed as *HSA_PACKET_HEADER_SCACQUIRE_FENCE_SCOPE*.

enumerator **HSA_PACKET_HEADER_SCRELEASE_FENCE_SCOPE**

Release fence scope, The value of this sub-field determines the scope and type of the memory fence operation applied after kernel completion but before the packet is completed. A release fence makes any global segment or image data that was stored by any unit of execution that belonged to a dispatch that has completed the active phase on any queue of the same kernel agent visible in all the scopes specified by the release fence. The value of this sub-field must be one of *hsa_fence_scope_t*.

enumerator **HSA_PACKET_HEADER_RELEASE_FENCE_SCOPE**

Deprecated:

Renamed as *HSA_PACKET_HEADER_SCRELEASE_FENCE_SCOPE*.

enum **hsa_packet_header_width_t**

Width (in bits) of the sub-fields in *hsa_packet_header_t*.

Values:

enumerator **HSA_PACKET_HEADER_WIDTH_TYPE**

enumerator **HSA_PACKET_HEADER_WIDTH_BARRIER**

enumerator **HSA_PACKET_HEADER_WIDTH_SCACQUIRE_FENCE_SCOPE**

enumerator **HSA_PACKET_HEADER_WIDTH_ACQUIRE_FENCE_SCOPE**

Deprecated:

Use HSA_PACKET_HEADER_WIDTH_SCACQUIRE_FENCE_SCOPE.

enumerator **HSA_PACKET_HEADER_WIDTH_SCRELEASE_FENCE_SCOPE**

enumerator **HSA_PACKET_HEADER_WIDTH_RELEASE_FENCE_SCOPE**

Deprecated:

Use HSA_PACKET_HEADER_WIDTH_SCRELEASE_FENCE_SCOPE.

enum **hsa_kernel_dispatch_packet_setup_t**

Sub-fields of the kernel dispatch packet *setup* field. The offset (with respect to the address of *setup*) of a sub-field is identical to its enumeration constant. The width of each sub-field is determined by the corresponding value in *hsa_kernel_dispatch_packet_setup_width_t*. The offset and the width are expressed in bits.

Values:

enumerator **HSA_KERNEL_DISPATCH_PACKET_SETUP_DIMENSIONS**

Number of dimensions of the grid. Valid values are 1, 2, or 3.

enum **hsa_kernel_dispatch_packet_setup_width_t**

Width (in bits) of the sub-fields in *hsa_kernel_dispatch_packet_setup_t*.

Values:

enumerator **HSA_KERNEL_DISPATCH_PACKET_SETUP_WIDTH_DIMENSIONS**

enum **hsa_amd_packet_type_t**

AMD vendor specific packet type.

Values:

enumerator **HSA_AMD_PACKET_TYPE_BARRIER_VALUE**

Packet used by agents to delay processing of subsequent packets until a configurable condition is satisfied by an HSA signal. Only kernel dispatch queues created from AMD GPU Agents support this packet.

enumerator **HSA_AMD_PACKET_TYPE_AIE_ERT**

Packet used to send commands to an AIE agent's embedded runtime (ERT). The ERT is responsible for, among other things, handling dispatches. Only queues created on AIE agents support this packet.

enum **hsa_amd_aie_ert_state**

State of an AIE ERT command.

Values:

enumerator **HSA_AMD_AIE_ERT_STATE_NEW**

Set by the host before submitting a command to the scheduler.

enumerator **HSA_AMD_AIE_ERT_STATE_QUEUED**

Internal scheduler state.

enumerator **HSA_AMD_AIE_ERT_STATE_RUNNING**

Internal scheduler state.

enumerator **HSA_AMD_AIE_ERT_STATE_COMPLETED**

Set by the scheduler when a command completes.

enumerator **HSA_AMD_AIE_ERT_STATE_ERROR**

Set by the scheduler if a command failed.

enumerator **HSA_AMD_AIE_ERT_STATE_ABORT**

Set by the scheduler if a command aborted.

enumerator **HSA_AMD_AIE_ERT_STATE_SUBMITTED**

Internal scheduler state.

enumerator **HSA_AMD_AIE_ERT_STATE_TIMEOUT**

Set by the scheduler on a timeout and reset.

enumerator **HSA_AMD_AIE_ERT_STATE_NORESPONSE**

Set by the scheduler on a timeout and fail to reset.

enumerator **HSA_AMD_AIE_ERT_STATE_SKERROR**

enumerator **HSA_AMD_AIE_ERT_STATE_SKCRASHED**

enumerator **HSA_AMD_AIE_ERT_STATE_MAX**

enum **hsa_amd_aie_ert_cmd_opcode_t**

Opcode types for HSA AIE ERT commands.

Values:

enumerator **HSA_AMD_AIE_ERT_START_CU**

Start a workgroup on a compute unit (CU).

enumerator **HSA_AMD_AIE_ERT_START_KERNEL**

Currently aliased to HSA_AMD_AIE_ERT_START_CU.

enumerator **HSA_AMD_AIE_ERT_CONFIGURE**

Configure command scheduler.

enumerator **HSA_AMD_AIE_ERT_EXIT**

enumerator **HSA_AMD_AIE_ERT_ABORT**

enumerator **HSA_AMD_AIE_ERT_EXEC_WRITE**

Execute a specified CU after writing.

enumerator **HSA_AMD_AIE_ERT_CU_STAT**

Get stats about a CU's execution.

enumerator **HSA_AMD_AIE_ERT_START_COPYBO**

Start KDMA CU or P2P.

enumerator **HSA_AMD_AIE_ERT_SK_CONFIG**

Configure a soft kernel.

enumerator **HSA_AMD_AIE_ERT_SK_START**

Start a soft kernel.

enumerator **HSA_AMD_AIE_ERT_SK_UNCONFIG**

Unconfigure a soft kernel.

enumerator **HSA_AMD_AIE_ERT_INIT_CU**

Initialize a CU.

enumerator **HSA_AMD_AIE_ERT_START_FA**

enumerator **HSA_AMD_AIE_ERT_CLK_CALIB**

enumerator **HSA_AMD_AIE_ERT_MB_VALIDATE**

enumerator **HSA_AMD_AIE_ERT_START_KEY_VAL**

Same as HSA_AMD_AIE_ERT_START_CU but with a key-value pair.

enumerator **HSA_AMD_AIE_ERT_ACCESS_TEST_C**

enumerator **HSA_AMD_AIE_ERT_ACCESS_TEST**

enumerator **HSA_AMD_AIE_ERT_START_DPU**

Instruction buffer command format.

enumerator **HSA_AMD_AIE_ERT_CMD_CHAIN**

Command chain.

enumerator **HSA_AMD_AIE_ERT_START_NPU**

Instruction buffer command format on NPU.

enumerator **HSA_AMD_AIE_ERT_START_NPU_PREEMPT**

Instruction buffer command with pre-emption format on the NPU.

enum **hsa_amd_aie_ert_cmd_type_t**

Command types for HSA AMD AIE ERT.

Values:

enumerator **HSA_AMD_AIE_ERT_CMD_TYPE_DEFAULT**

Default command type.

enumerator **HSA_AMD_AIE_ERT_CMD_TYPE_KDS_LOCAL**

Command processed by kernel domain scheduler (KDS) locally.

enumerator **HSA_AMD_AIE_ERT_CMD_TYPE_CTRL**

Control command uses reserved command queue slot.

enumerator **HSA_AMD_AIE_ERT_CMD_TYPE_CU**

Control command uses reserved command queue slot.

enumerator **HSA_AMD_AIE_ERT_CMD_TYPE_SCU**

CU command.

typedef struct *hsa_kernel_dispatch_packet_s* **hsa_kernel_dispatch_packet_t**

AQL kernel dispatch packet.

typedef struct *hsa_agent_dispatch_packet_s* **hsa_agent_dispatch_packet_t**

Agent dispatch packet.

typedef struct *hsa_barrier_and_packet_s* **hsa_barrier_and_packet_t**

Barrier-AND packet.

typedef struct *hsa_barrier_or_packet_s* **hsa_barrier_or_packet_t**

Barrier-OR packet.

typedef uint32_t **hsa_signal_condition32_t**

A fixed-size type used to represent *hsa_signal_condition_t* constants.

typedef uint8_t **hsa_amd_packet_type8_t**

A fixed-size type used to represent *hsa_amd_packet_type_t* constants.

typedef struct *hsa_amd_packet_header_s* **hsa_amd_vendor_packet_header_t**

AMD vendor specific AQL packet header.

typedef struct *hsa_amd_barrier_value_packet_s* **hsa_amd_barrier_value_packet_t**

AMD barrier value packet. Halts packet processing and waits for (signal_value & ::mask) ::cond ::value to be satisfied, where signal_value is the value of the signal ::signal.

typedef struct *hsa_amd_aie_ert_start_kernel_header_s* **hsa_amd_aie_ert_start_kernel_header_t**

Format for start kernel packet header.

typedef struct *hsa_amd_aie_ert_start_kernel_data_s* **hsa_amd_aie_ert_start_kernel_data_t**

Payload data for AIE ERT start kernel packets (i.e., when the opcode is HSA_AMD_AIE_ERT_START_KERNEL).

typedef struct *hsa_amd_aie_ert_command_chain_data_s* **hsa_amd_aie_ert_command_chain_data_t**

Payload data for AIE ERT command chain packets (i.e., when the opcode is HSA_AMD_AIE_ERT_CMD_CHAIN). A command chain is a buffer of commands parsed by the ERT.

typedef struct *hsa_amd_aie_ert_packet_s* **hsa_amd_aie_ert_packet_t**

AMD AIE ERT packet. Used for sending a command to an AIE agent.

static inline void **hsa_flag_set64**(uint8_t *value, uint32_t bit)

Macro to set a flag within uint8_t[8] types.

static inline bool **hsa_flag_isset64**(uint8_t *value, uint32_t bit)

Macro to determine whether a flag is set within uint8_t[8] types.

struct **hsa_kernel_dispatch_packet_s**

#include <*hsa.h*> AQL kernel dispatch packet.

struct **hsa_agent_dispatch_packet_s**

#include <*hsa.h*> Agent dispatch packet.

struct **hsa_barrier_and_packet_s**

#include <*hsa.h*> Barrier-AND packet.

struct **hsa_barrier_or_packet_s**

#include <*hsa.h*> Barrier-OR packet.

struct **hsa_amd_packet_header_s**

#include <*hsa_ext_amd.h*> AMD vendor specific AQL packet header.

struct **hsa_amd_barrier_value_packet_s**

#include <*hsa_ext_amd.h*> AMD barrier value packet. Halts packet processing and waits for (signal_value & ::mask) ::cond ::value to be satisfied, where signal_value is the value of the signal ::signal.

struct **hsa_amd_aie_ert_start_kernel_header_s**

#include <*hsa_ext_amd.h*> Format for start kernel packet header.

struct **hsa_amd_aie_ert_start_kernel_data_s**

#include <*hsa_ext_amd.h*> Payload data for AIE ERT start kernel packets (i.e., when the opcode is HSA_AMD_AIE_ERT_START_KERNEL).

struct **hsa_amd_aie_ert_command_chain_data_s**

#include <*hsa_ext_amd.h*> Payload data for AIE ERT command chain packets (i.e., when the opcode is HSA_AMD_AIE_ERT_CMD_CHAIN). A command chain is a buffer of commands parsed by the ERT.

struct **hsa_amd_aie_ert_packet_s**

#include <*hsa_ext_amd.h*> AMD AIE ERT packet. Used for sending a command to an AIE agent.

4.2 Code objects [DEPRECATED]

enum **hsa_code_object_type_t**

Code object type.

Deprecated:

Values:

enumerator **HSA_CODE_OBJECT_TYPE_PROGRAM**

Produces code object that contains ISA for all kernels and indirect functions in HSA source.

enum **hsa_code_object_info_t**

Code object attributes.

Deprecated:

Values:

enumerator **HSA_CODE_OBJECT_INFO_VERSION**

The version of the code object. The type of this attribute is a NUL-terminated char[64]. The name must be at most 63 characters long (not including the NUL terminator) and all array elements not used for the name must be NUL.

enumerator **HSA_CODE_OBJECT_INFO_TYPE**

Type of code object. The type of this attribute is *hsa_code_object_type_t*.

enumerator **HSA_CODE_OBJECT_INFO_ISA**

Instruction set architecture this code object is produced for. The type of this attribute is *hsa_isa_t*.

enumerator **HSA_CODE_OBJECT_INFO_MACHINE_MODEL**

Machine model this code object is produced for. The type of this attribute is *hsa_machine_model_t*.

enumerator **HSA_CODE_OBJECT_INFO_PROFILE**

Profile this code object is produced for. The type of this attribute is *hsa_profile_t*.

enumerator **HSA_CODE_OBJECT_INFO_DEFAULT_FLOAT_ROUNDING_MODE**

Default floating-point rounding mode used when the code object is produced. The type of this attribute is *hsa_default_float_rounding_mode_t*.

enum **hsa_code_symbol_info_t**

Code object symbol attributes.

Deprecated:

Values:

enumerator **HSA_CODE_SYMBOL_INFO_TYPE**

The type of the symbol. The type of this attribute is *hsa_symbol_kind_t*.

enumerator **HSA_CODE_SYMBOL_INFO_NAME_LENGTH**

The length of the symbol name in bytes, not including the NUL terminator. The type of this attribute is uint32_t.

enumerator **HSA_CODE_SYMBOL_INFO_NAME**

The name of the symbol. The type of this attribute is character array with the length equal to the value of *HSA_CODE_SYMBOL_INFO_NAME_LENGTH* attribute.

enumerator **HSA_CODE_SYMBOL_INFO_MODULE_NAME_LENGTH**

The length of the module name in bytes (not including the NUL terminator) to which this symbol belongs if this symbol has module linkage, otherwise 0 is returned. The type of this attribute is uint32_t.

enumerator **HSA_CODE_SYMBOL_INFO_MODULE_NAME**

The module name to which this symbol belongs if this symbol has module linkage, otherwise an empty string is returned. The type of this attribute is character array with the length equal to the value of *HSA_CODE_SYMBOL_INFO_MODULE_NAME_LENGTH* attribute.

enumerator **HSA_CODE_SYMBOL_INFO_LINKAGE**

The linkage kind of the symbol. The type of this attribute is *hsa_symbol_linkage_t*.

enumerator **HSA_CODE_SYMBOL_INFO_IS_DEFINITION**

Indicates whether the symbol corresponds to a definition. The type of this attribute is bool.

enumerator **HSA_CODE_SYMBOL_INFO_VARIABLE_ALLOCATION**

The allocation kind of the variable. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is *hsa_variable_allocation_t*.

enumerator **HSA_CODE_SYMBOL_INFO_VARIABLE_SEGMENT**

The segment kind of the variable. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is *hsa_variable_segment_t*.

enumerator **HSA_CODE_SYMBOL_INFO_VARIABLE_ALIGNMENT**

Alignment of the symbol in memory. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is uint32_t.

The current alignment of the variable in memory may be greater than the value specified in the source program variable declaration.

enumerator **HSA_CODE_SYMBOL_INFO_VARIABLE_SIZE**

Size of the variable. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is uint32_t.

A size of 0 is returned if the variable is an external variable and has an unknown dimension.

enumerator **HSA_CODE_SYMBOL_INFO_VARIABLE_IS_CONST**

Indicates whether the variable is constant. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is bool.

enumerator **HSA_CODE_SYMBOL_INFO_KERNEL_KERNARG_SEGMENT_SIZE**

Size of kernarg segment memory that is required to hold the values of the kernel arguments, in bytes. Must be a multiple of 16. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

enumerator **HSA_CODE_SYMBOL_INFO_KERNEL_KERNARG_SEGMENT_ALIGNMENT**

Alignment (in bytes) of the buffer used to pass arguments to the kernel, which is the maximum of 16 and the maximum alignment of any of the kernel arguments. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

enumerator **HSA_CODE_SYMBOL_INFO_KERNEL_GROUP_SEGMENT_SIZE**

Size of static group segment memory required by the kernel (per work-group), in bytes. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

The reported amount does not include any dynamically allocated group segment memory that may be requested by the application when a kernel is dispatched.

enumerator **HSA_CODE_SYMBOL_INFO_KERNEL_PRIVATE_SEGMENT_SIZE**

Size of static private, spill, and arg segment memory required by this kernel (per work-item), in bytes. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

If the value of *HSA_CODE_SYMBOL_INFO_KERNEL_DYNAMIC_CALLSTACK* is true, the kernel may use more private memory than the reported value, and the application must add the dynamic call stack usage to *private_segment_size* when populating a kernel dispatch packet.

enumerator **HSA_CODE_SYMBOL_INFO_KERNEL_DYNAMIC_CALLSTACK**

Dynamic callstack flag. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is bool.

If this flag is set (the value is true), the kernel uses a dynamically sized call stack. This can happen if recursive calls, calls to indirect functions, or the HSAIL alloca instruction are present in the kernel.

enumerator **HSA_CODE_SYMBOL_INFO_KERNEL_CALL_CONVENTION**

Call convention of the kernel. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

enumerator **HSA_CODE_SYMBOL_INFO_INDIRECT_FUNCTION_CALL_CONVENTION**

Call convention of the indirect function. The value of this attribute is undefined if the symbol is not an indirect function. The type of this attribute is uint32_t.

enumerator **HSA_CODE_SYMBOL_INFO_KERNEL_WAVEFRONT_SIZE**

Wavefront size used by the kernel. The value of this attribute is either 32 or 64. The type of this attribute is uint32_t.

typedef struct *hsa_code_object_s* **hsa_code_object_t**

Struct containing an opaque handle to a code object, which contains ISA for finalized kernels and indirect functions together with information about the global or readonly segment variables they reference.

Deprecated:

typedef struct *hsa_callback_data_s* **hsa_callback_data_t**

Application data handle that is passed to the serialization and deserialization functions.

Deprecated:

typedef struct *hsa_code_symbol_s* **hsa_code_symbol_t**

Code object symbol handle.

Deprecated:

The lifetime of a code object symbol matches that of the code object associated with it. An operation on a symbol whose associated code object has been destroyed results in undefined behavior.

hsa_status_t HSA_API HSA_DEPRECATED hsa_code_object_serialize (hsa_code_object_t code_object, hsa_status_t(*alloc_callback)(size_t size, hsa_callback_data_t data, void **address), hsa_callback_data_t callback_data, const char *options, void **serialized_code_object, size_t *serialized_code_object_size)

Serialize a code object. Can be used for offline finalization, install-time finalization, disk code caching, etc.

Deprecated:

Parameters

- **code_object** – **[in]** Code object.
- **alloc_callback** – **[in]** Callback function for memory allocation. Must not be NULL. The HSA runtime passes three arguments to the callback: the allocation size, the application data, and a pointer to a memory location where the application stores the allocation result. The HSA runtime invokes `alloc_callback` once to allocate a buffer that contains the serialized version of `code_object`. If the callback returns a status code other than *HSA_STATUS_SUCCESS*, this function returns the same code.
- **callback_data** – **[in]** Application data that is passed to `alloc_callback`. May be NULL.
- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.
- **serialized_code_object** – **[out]** Memory location where the HSA runtime stores a pointer to the serialized code object. Must not be NULL.
- **serialized_code_object_size** – **[out]** Memory location where the HSA runtime stores the size (in bytes) of `serialized_code_object`. The returned value matches the allocation size passed by the HSA runtime to `alloc_callback`. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT** – `code_object` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `alloc_callback`, `serialized_code_object`, or `serialized_code_object_size` are NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_code_object_deserialize (void *serialized_code_object, size_t serialized_code_object_size, const char *options, hsa_code_object_t *code_object)

Deserialize a code object.

Deprecated:

Parameters

- **serialized_code_object** – **[in]** A serialized code object. Must not be NULL.

- **serialized_code_object_size** – **[in]** The size (in bytes) of `serialized_code_object`. Must not be 0.
- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.
- **code_object** – **[out]** Memory location where the HSA runtime stores the deserialized code object.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `serialized_code_object`, or `code_object` are NULL, or `serialized_code_object_size` is 0.

hsa_status_t HSA_API HSA_DEPRECATED hsa_code_object_destroy (hsa_code_object_t code_object)

Destroy a code object.

Deprecated:

The lifetime of a code object must exceed that of any executable where it has been loaded. If an executable that loaded `code_object` has not been destroyed, the behavior is undefined.

Parameters

code_object – **[in]** Code object. The handle becomes invalid after it has been destroyed.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT** – `code_object` is invalid.

hsa_status_t HSA_API HSA_DEPRECATED hsa_code_object_get_info (hsa_code_object_t code_object, hsa_code_object_info_t attribute, void *value)

Get the current value of an attribute for a given code object.

Deprecated:

Parameters

- **code_object** – **[in]** Code object.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT** – `code_object` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid code object attribute, or `value` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_executable_load_code_object (hsa_executable_t executable, hsa_agent_t agent, hsa_code_object_t code_object, const char *options)

Load code object into the executable.

Deprecated:

Every global or readonly variable that is external must be defined before loading the code object. An internal global or readonly variable is allocated once the code object, that is being loaded, references this variable and this variable is not allocated.

Any module linkage declaration must have been defined either by a define variable or by loading a code object that has a symbol with module linkage definition.

Parameters

- **executable** – **[in]** Executable.
- **agent** – **[in]** Agent to load code object for. The agent must support the default floating-point rounding mode used by `code_object`.
- **code_object** – **[in]** Code object to load. The lifetime of the code object must exceed that of the executable: if `code_object` is destroyed before `executable`, the behavior is undefined.
- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT** – `code_object` is invalid.
- **HSA_STATUS_ERROR_INCOMPATIBLE_ARGUMENTS** – `agent` is not compatible with `code_object` (for example, `agent` does not support the default floating-point rounding mode specified by `code_object`), or `code_object` is not compatible with `executable` (for example, `code_object` and `executable` have different machine models or profiles).
- **HSA_STATUS_ERROR_FROZEN_EXECUTABLE** – `executable` is frozen.

hsa_status_t HSA_API HSA_DEPRECATED hsa_code_object_get_symbol (hsa_code_object_t code_object, const char *symbol_name, hsa_code_symbol_t *symbol)

Get the symbol handle within a code object for a given a symbol name.

Deprecated:

Parameters

- **code_object** – **[in]** Code object.
- **symbol_name** – **[in]** Symbol name.
- **symbol** – **[out]** Memory location where the HSA runtime stores the symbol handle.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT** – code_object is invalid.
- **HSA_STATUS_ERROR_INVALID_SYMBOL_NAME** – There is no symbol with a name that matches symbol_name.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – symbol_name is NULL, or symbol is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_code_object_get_symbol_from_name (hsa_code_object_t code_object const char *module_name, const char *symbol_name, hsa_code_symbol_t *symbol)

Get the symbol handle within a code object for a given a symbol name.

Deprecated:

Parameters

- **code_object** – **[in]** Code object.
- **module_name** – **[in]** Module name. Must be NULL if the symbol has program linkage.
- **symbol_name** – **[in]** Symbol name.
- **symbol** – **[out]** Memory location where the HSA runtime stores the symbol handle.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT** – code_object is invalid.
- **HSA_STATUS_ERROR_INVALID_SYMBOL_NAME** – There is no symbol with a name that matches symbol_name.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – symbol_name is NULL, or symbol is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_code_symbol_get_info (hsa_code_symbol_t code_symbol, hsa_code_symbol_info_t attribute, void *value)

Get the current value of an attribute for a given code symbol.

Deprecated:

Parameters

- **code_symbol** – **[in]** Code symbol.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_CODE_SYMBOL** – The code symbol is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid code symbol attribute, or `value` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_code_object_iterate_symbols (hsa_code_object_t code_object, hsa_status_t(*callback)(hsa_code_object_t code_object, hsa_code_symbol_t symbol, void *data), void *data)

Iterate over the symbols in a code object, and invoke an application-defined callback on every iteration.

Deprecated:

Parameters

- **code_object** – **[in]** Code object.
- **callback** – **[in]** Callback to be invoked once per code object symbol. The HSA runtime passes three arguments to the callback: the code object, a symbol, and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and *hsa_code_object_iterate_symbols* returns that status value.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT** – `code_object` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

struct **hsa_code_object_s**

#include <hsa.h> Struct containing an opaque handle to a code object, which contains ISA for finalized kernels and indirect functions together with information about the global or readonly segment variables they reference.

Deprecated:

struct **hsa_callback_data_s**

#include <hsa.h> Application data handle that is passed to the serialization and deserialization functions.

Deprecated:

struct **hsa_code_symbol_s**

#include <hsa.h> Code object symbol handle.

Deprecated:

The lifetime of a code object symbol matches that of the code object associated with it. An operation on a symbol whose associated code object has been destroyed results in undefined behavior.

4.3 Executable

enum **hsa_executable_state_t**

Executable state.

Values:

enumerator **HSA_EXECUTABLE_STATE_UNFROZEN**

Executable state, which allows the user to load code objects and define external variables. Variable addresses, kernel code handles, and indirect function code handles are not available in query operations until the executable is frozen (zero always returned).

enumerator **HSA_EXECUTABLE_STATE_FROZEN**

Executable state, which allows the user to query variable addresses, kernel code handles, and indirect function code handles using query operations. Loading new code objects, as well as defining external variables, is not allowed in this state.

enum **hsa_executable_info_t**

Executable attributes.

Values:

enumerator **HSA_EXECUTABLE_INFO_PROFILE**

Profile this executable is created for. The type of this attribute is *hsa_profile_t*.

enumerator **HSA_EXECUTABLE_INFO_STATE**

Executable state. The type of this attribute is *hsa_executable_state_t*.

enumerator **HSA_EXECUTABLE_INFO_DEFAULT_FLOAT_ROUNDING_MODE**

Default floating-point rounding mode specified when executable was created. The type of this attribute is *hsa_default_float_rounding_mode_t*.

enum **hsa_symbol_kind_t**

Symbol type.

Values:

enumerator **HSA_SYMBOL_KIND_VARIABLE**

Variable.

enumerator **HSA_SYMBOL_KIND_KERNEL**

Kernel.

enumerator **HSA_SYMBOL_KIND_INDIRECT_FUNCTION**

Indirect function.

enum **hsa_symbol_linkage_t**

Linkage type of a symbol.

Values:

enumerator **HSA_SYMBOL_LINKAGE_MODULE**

Module linkage.

enumerator **HSA_SYMBOL_LINKAGE_PROGRAM**

Program linkage.

enum **hsa_variable_allocation_t**

Allocation type of a variable.

Values:

enumerator **HSA_VARIABLE_ALLOCATION_AGENT**

Agent allocation.

enumerator **HSA_VARIABLE_ALLOCATION_PROGRAM**

Program allocation.

enum **hsa_variable_segment_t**

Memory segment associated with a variable.

Values:

enumerator **HSA_VARIABLE_SEGMENT_GLOBAL**

Global memory segment.

enumerator **HSA_VARIABLE_SEGMENT_READONLY**

Readonly memory segment.

enum **hsa_executable_symbol_info_t**

Executable symbol attributes.

Values:

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_TYPE**

The kind of the symbol. The type of this attribute is *hsa_symbol_kind_t*.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_NAME_LENGTH**

The length of the symbol name in bytes, not including the NUL terminator. The type of this attribute is uint32_t.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_NAME**

The name of the symbol. The type of this attribute is character array with the length equal to the value of *HSA_EXECUTABLE_SYMBOL_INFO_NAME_LENGTH* attribute.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_MODULE_NAME_LENGTH**

Deprecated:

The length of the module name in bytes (not including the NUL terminator) to which this symbol belongs if this symbol has module linkage, otherwise 0 is returned. The type of this attribute is uint32_t.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_MODULE_NAME**

Deprecated:

The module name to which this symbol belongs if this symbol has module linkage, otherwise an empty string is returned. The type of this attribute is character array with the length equal to the value of *HSA_EXECUTABLE_SYMBOL_INFO_MODULE_NAME_LENGTH* attribute.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_AGENT**

Deprecated:

Agent associated with this symbol. If the symbol is a variable, the value of this attribute is only defined if *HSA_EXECUTABLE_SYMBOL_INFO_VARIABLE_ALLOCATION* is *HSA_VARIABLE_ALLOCATION_AGENT*. The type of this attribute is hsa_agent_t.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_VARIABLE_ADDRESS**

The address of the variable. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is uint64_t.

If executable's state is *HSA_EXECUTABLE_STATE_UNFROZEN*, then 0 is returned.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_LINKAGE**

The linkage kind of the symbol. The type of this attribute is *hsa_symbol_linkage_t*.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_IS_DEFINITION**

Indicates whether the symbol corresponds to a definition. The type of this attribute is bool.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_VARIABLE_ALLOCATION**

Deprecated:

The allocation kind of the variable. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is *hsa_variable_allocation_t*.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_VARIABLE_SEGMENT**

Deprecated:

The segment kind of the variable. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is *hsa_variable_segment_t*.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_VARIABLE_ALIGNMENT**

Deprecated:

Alignment of the symbol in memory. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is uint32_t.

The current alignment of the variable in memory may be greater than the value specified in the source program variable declaration.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_VARIABLE_SIZE**

Deprecated:

Size of the variable. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is uint32_t.

A value of 0 is returned if the variable is an external variable and has an unknown dimension.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_VARIABLE_IS_CONST**

Deprecated:

Indicates whether the variable is constant. The value of this attribute is undefined if the symbol is not a variable. The type of this attribute is bool.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_KERNEL_OBJECT**

Kernel object handle, used in the kernel dispatch packet. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint64_t.

If the state of the executable is *HSA_EXECUTABLE_STATE_UNFROZEN*, then 0 is returned.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_KERNEL_KERNARG_SEGMENT_SIZE**

Size of kernarg segment memory that is required to hold the values of the kernel arguments, in bytes. Must be a multiple of 16. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_KERNEL_KERNARG_SEGMENT_ALIGNMENT**

Alignment (in bytes) of the buffer used to pass arguments to the kernel, which is the maximum of 16 and the maximum alignment of any of the kernel arguments. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_KERNEL_GROUP_SEGMENT_SIZE**

Size of static group segment memory required by the kernel (per work-group), in bytes. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

The reported amount does not include any dynamically allocated group segment memory that may be requested by the application when a kernel is dispatched.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_KERNEL_PRIVATE_SEGMENT_SIZE**

Size of static private, spill, and arg segment memory required by this kernel (per work-item), in bytes. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

If the value of *HSA_EXECUTABLE_SYMBOL_INFO_KERNEL_DYNAMIC_CALLSTACK* is true, the kernel may use more private memory than the reported value, and the application must add the dynamic call stack usage to *private_segment_size* when populating a kernel dispatch packet.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_KERNEL_DYNAMIC_CALLSTACK**

Dynamic callstack flag. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is bool.

If this flag is set (the value is true), the kernel uses a dynamically sized call stack. This can happen if recursive calls, calls to indirect functions, or the HSAIL alloca instruction are present in the kernel.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_KERNEL_CALL_CONVENTION**

Deprecated:

Call convention of the kernel. The value of this attribute is undefined if the symbol is not a kernel. The type of this attribute is uint32_t.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_INDIRECT_FUNCTION_OBJECT**

Indirect function object handle. The value of this attribute is undefined if the symbol is not an indirect function, or the associated agent does not support the Full Profile. The type of this attribute depends on the machine model: the type is uint32_t for small machine model, and uint64_t for large model.

If the state of the executable is *HSA_EXECUTABLE_STATE_UNFROZEN*, then 0 is returned.

enumerator **HSA_EXECUTABLE_SYMBOL_INFO_INDIRECT_FUNCTION_CALL_CONVENTION**

Deprecated:

Call convention of the indirect function. The value of this attribute is undefined if the symbol is not an indirect function, or the associated agent does not support the Full Profile. The type of this attribute is uint32_t.

typedef struct *hsa_code_object_reader_s* **hsa_code_object_reader_t**

Code object reader handle. A code object reader is used to load a code object from file (when created using *hsa_code_object_reader_create_from_file*), or from memory (if created using *hsa_code_object_reader_create_from_memory*).

typedef struct *hsa_executable_s* **hsa_executable_t**

Struct containing an opaque handle to an executable, which contains ISA for finalized kernels and indirect functions together with the allocated global or readonly segment variables they reference.

typedef struct *hsa_loaded_code_object_s* **hsa_loaded_code_object_t**

Loaded code object handle.

typedef struct *hsa_executable_symbol_s* **hsa_executable_symbol_t**

Executable symbol handle.

The lifetime of an executable object symbol matches that of the executable associated with it. An operation on a symbol whose associated executable has been destroyed results in undefined behavior.

hsa_status_t HSA_API hsa_code_object_reader_create_from_file (hsa_file_t file, hsa_code_object_reader_t *code_object_reader)

Create a code object reader to operate on a file.

The file is owned and managed by the application; the lifetime of the file descriptor must exceed that of any associated code object reader.

Parameters

- **file** – **[in]** File descriptor. The file must have been opened by application with at least read permissions prior calling this function. The file must contain a vendor-specific code object.
- **code_object_reader** – **[out]** Memory location to store the newly created code object reader handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_FILE** – `file` is invalid.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `code_object_reader` is NULL.

hsa_status_t HSA_API hsa_code_object_reader_create_from_memory (const void *code_object, size_t size, hsa_code_object_reader_t *code_object_reader)

Create a code object reader to operate on memory.

Parameters

- **code_object** – **[in]** Memory buffer that contains a vendor-specific code object. The buffer is owned and managed by the application; the lifetime of the buffer must exceed that of any associated code object reader.

- **size** – **[in]** Size of the buffer pointed to by code_object. Must not be 0.
- **code_object_reader** – **[out]** Memory location to store newly created code object reader handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – code_object is NULL, size is zero, or code_object_reader is NULL.

hsa_status_t HSA_API hsa_code_object_reader_destroy (hsa_code_object_reader_t code_object_reader)

Destroy a code object reader.

The code object reader handle becomes invalid after completion of this function. Any file or memory used to create the code object read is not closed, removed, or deallocated by this function.

Parameters

code_object_reader – **[in]** Code object reader to destroy.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT_READER** – code_object_reader is invalid.

hsa_status_t HSA_API HSA_DEPRECATED hsa_executable_create (hsa_profile_t profile, hsa_executable_state_t executable_state, const char *options, hsa_executable_t *executable)

Create an empty executable.

Deprecated:

Use *hsa_executable_create_alt* instead, which allows the application to specify the default floating-point rounding mode of the executable and assumes an unfrozen initial state.

Parameters

- **profile** – **[in]** Profile used in the executable.
- **executable_state** – **[in]** Executable state. If the state is *HSA_EXECUTABLE_STATE_FROZEN*, the resulting executable is useless because no code objects can be loaded, and no variables can be defined.
- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.
- **executable** – **[out]** Memory location where the HSA runtime stores the newly created executable handle.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `profile` is invalid, or `executable` is NULL.

hsa_status_t HSA_API hsa_executable_create_alt (hsa_profile_t profile, hsa_default_float_rounding_mode_t default_float_rounding_mode, const char *options, hsa_executable_t *executable)

Create an empty executable.

Parameters

- **profile** – **[in]** Profile used in the executable.
- **default_float_rounding_mode** – **[in]** Default floating-point rounding mode used in the executable. Allowed rounding modes are near and zero (default is not allowed).
- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.
- **executable** – **[out]** Memory location where the HSA runtime stores newly created executable handle. The initial state of the executable is *HSA_EXECUTABLE_STATE_UNFROZEN*.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `profile` is invalid, or `executable` is NULL.

hsa_status_t HSA_API hsa_executable_destroy (hsa_executable_t executable)

Destroy an executable.

An executable handle becomes invalid after the executable has been destroyed. Code object handles that were loaded into this executable are still valid after the executable has been destroyed, and can be used as intended. Resources allocated outside and associated with this executable (such as external global or readonly variables) can be released after the executable has been destroyed.

Executable should not be destroyed while kernels are in flight.

Parameters

executable – **[in]** Executable.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.

hsa_status_t HSA_API hsa_executable_load_program_code_object (hsa_executable_t executable, hsa_code_object_reader_t code_object_reader, const char *options, hsa_loaded_code_object_t *loaded_code_object)

Load a program code object into an executable.

A program code object contains information about resources that are accessible by all kernel agents that run the executable, and can be loaded at most once into an executable.

If the program code object uses extensions, the implementation must support them for this operation to return successfully.

Parameters

- **executable** – **[in]** Executable.
- **code_object_reader** – **[in]** A code object reader that holds the program code object to load. If a code object reader is destroyed before all the associated executables are destroyed, the behavior is undefined.
- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.
- **loaded_code_object** – **[out]** Pointer to a memory location where the HSA runtime stores the loaded code object handle. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_FROZEN_EXECUTABLE** – The executable is frozen.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT_READER** – code_object_reader is invalid.
- **HSA_STATUS_ERROR_INCOMPATIBLE_ARGUMENTS** – The program code object is not compatible with the executable or the implementation (for example, the code object uses an extension that is not supported by the implementation).

hsa_status_t HSA_API hsa_executable_load_agent_code_object (hsa_executable_t executable, hsa_agent_t agent, hsa_code_object_reader_t code_object_reader, const char *options, hsa_loaded_code_object_t *loaded_code_object)

Load an agent code object into an executable.

The agent code object contains all defined agent allocation variables, functions, indirect functions, and kernels in a given program for a given instruction set architecture.

Any module linkage declaration must have been defined either by a define variable or by loading a code object that has a symbol with module linkage definition.

The default floating-point rounding mode of the code object associated with `code_object_reader` must match that of the executable (*HSA_EXECUTABLE_INFO_DEFAULT_FLOAT_ROUNDING_MODE*), or be default (in which case the value of *HSA_EXECUTABLE_INFO_DEFAULT_FLOAT_ROUNDING_MODE* is used). If the agent code object uses extensions, the implementation and the agent must support them for this operation to return successfully.

Parameters

- **executable** – **[in]** Executable.
- **agent** – **[in]** Agent to load code object for. A code object can be loaded into an executable at most once for a given agent. The instruction set architecture of the code object must be supported by the agent.
- **code_object_reader** – **[in]** A code object reader that holds the code object to load. If a code object reader is destroyed before all the associated executables are destroyed, the behavior is undefined.
- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.
- **loaded_code_object** – **[out]** Pointer to a memory location where the HSA runtime stores the loaded code object handle. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_FROZEN_EXECUTABLE** – The executable is frozen.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_CODE_OBJECT_READER** – `code_object_reader` is invalid.
- **HSA_STATUS_ERROR_INCOMPATIBLE_ARGUMENTS** – The code object read by `code_object_reader` is not compatible with the agent (for example, the agent does not support the instruction set architecture of the code object), the executable (for example, there is a default floating-point mode mismatch between the two), or the implementation.

hsa_status_t HSA_API hsa_executable_freeze (hsa_executable_t executable, const char *options)

Freeze the executable.

No modifications to executable can be made after freezing: no code objects can be loaded to the executable, and no external variables can be defined. Freezing the executable does not prevent querying the executable's attributes. The application must define all the external variables in an executable before freezing it.

Parameters

- **executable** – **[in]** Executable.

- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_VARIABLE_UNDEFINED** – One or more variables are undefined in the executable.
- **HSA_STATUS_ERROR_FROZEN_EXECUTABLE** – `executable` is already frozen.

hsa_status_t HSA_API hsa_executable_get_info (hsa_executable_t executable, hsa_executable_info_t attribute, void *value)

Get the current value of an attribute for a given executable.

Parameters

- **executable** – **[in]** Executable.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid executable attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_executable_global_variable_define (hsa_executable_t executable, const char *variable_name, void *address)

Define an external global variable with program allocation.

This function allows the application to provide the definition of a variable in the global segment memory with program allocation. The variable must be defined before loading a code object into an executable. In addition, code objects loaded must not define the variable.

Parameters

- **executable** – **[in]** Executable. Must not be in frozen state.
- **variable_name** – **[in]** Name of the variable. The Programmer's Reference Manual describes the standard name mangling scheme.
- **address** – **[in]** Address where the variable is defined. This address must be in global memory and can be read and written by any agent in the system. The application cannot deallocate the buffer pointed by `address` before `executable` is destroyed.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_VARIABLE_ALREADY_DEFINED** – The variable is already defined.
- **HSA_STATUS_ERROR_INVALID_SYMBOL_NAME** – There is no variable with the `variable_name`.
- **HSA_STATUS_ERROR_FROZEN_EXECUTABLE** – `executable` is frozen.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `variable_name` is NULL.

hsa_status_t HSA_API hsa_executable_agent_global_variable_define (hsa_executable_t executable, hsa_agent_t agent, const char *variable_name, void *address)

Define an external global variable with agent allocation.

This function allows the application to provide the definition of a variable in the global segment memory with agent allocation. The variable must be defined before loading a code object into an executable. In addition, code objects loaded must not define the variable.

Parameters

- **executable** – **[in]** Executable. Must not be in frozen state.
- **agent** – **[in]** Agent for which the variable is being defined.
- **variable_name** – **[in]** Name of the variable. The Programmer's Reference Manual describes the standard name mangling scheme.
- **address** – **[in]** Address where the variable is defined. This address must have been previously allocated using *hsa_memory_allocate* in a global region that is only visible to `agent`. The application cannot deallocate the buffer pointed by `address` before `executable` is destroyed.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_INVALID_AGENT** – `agent` is invalid.
- **HSA_STATUS_ERROR_VARIABLE_ALREADY_DEFINED** – The variable is already defined.
- **HSA_STATUS_ERROR_INVALID_SYMBOL_NAME** – There is no variable with the `variable_name`.
- **HSA_STATUS_ERROR_FROZEN_EXECUTABLE** – `executable` is frozen.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `variable_name` is NULL.

hsa_status_t HSA_API hsa_executable_readonly_variable_define (hsa_executable_t executable, hsa_agent_t agent, const char *variable_name, void *address)

Define an external readonly variable.

This function allows the application to provide the definition of a variable in the readonly segment memory. The variable must be defined before loading a code object into an executable. In addition, code objects loaded must not define the variable.

Parameters

- **executable** – **[in]** Executable. Must not be in frozen state.
- **agent** – **[in]** Agent for which the variable is being defined.
- **variable_name** – **[in]** Name of the variable. The Programmer's Reference Manual describes the standard name mangling scheme.
- **address** – **[in]** Address where the variable is defined. This address must have been previously allocated using *hsa_memory_allocate* in a readonly region associated with `agent`. The application cannot deallocate the buffer pointed by `address` before `executable` is destroyed.
- **address** – **[in]** Address where the variable is defined. The buffer pointed by `address` is owned by the application, and cannot be deallocated before `executable` is destroyed.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – Executable is invalid.
- **HSA_STATUS_ERROR_INVALID_AGENT** – `agent` is invalid.
- **HSA_STATUS_ERROR_VARIABLE_ALREADY_DEFINED** – The variable is already defined.
- **HSA_STATUS_ERROR_INVALID_SYMBOL_NAME** – There is no variable with the `variable_name`.
- **HSA_STATUS_ERROR_FROZEN_EXECUTABLE** – `executable` is frozen.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `variable_name` is NULL.

hsa_status_t HSA_API hsa_executable_validate (hsa_executable_t executable, uint32_t *result)

Validate an executable. Checks that all code objects have matching machine model, profile, and default floating-point rounding mode. Checks that all declarations have definitions. Checks declaration-definition compatibility (see the HSA Programming Reference Manual for compatibility rules). Invoking this function is equivalent to invoking *hsa_executable_validate_alt* with no options.

Parameters

- **executable** – **[in]** Executable. Must be in frozen state.
- **result** – **[out]** Memory location where the HSA runtime stores the validation result. If the executable passes validation, the result is 0.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – `executable` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `result` is NULL.

hsa_status_t HSA_API hsa_executable_validate_alt (hsa_executable_t executable, const char *options, uint32_t *result)

Validate an executable. Checks that all code objects have matching machine model, profile, and default floating-point rounding mode. Checks that all declarations have definitions. Checks declaration-definition compatibility (see the HSA Programming Reference Manual for compatibility rules).

Parameters

- **executable** – **[in]** Executable. Must be in frozen state.
- **options** – **[in]** Standard and vendor-specific options. Unknown options are ignored. A standard option begins with the "-hsa_" prefix. Options beginning with the "-hsa_ext_<extension_name>_" prefix are reserved for extensions. A vendor-specific option begins with the "-<vendor_name>_" prefix. Must be a NUL-terminated string. May be NULL.
- **result** – **[out]** Memory location where the HSA runtime stores the validation result. If the executable passes validation, the result is 0.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – `executable` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `result` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_executable_get_symbol (hsa_executable_t executable, const char *module_name, const char *symbol_name, hsa_agent_t agent, int32_t call_convention, hsa_executable_symbol_t *symbol)

Get the symbol handle for a given a symbol name.

Deprecated:

Use *hsa_executable_get_symbol_by_name* instead.

Parameters

- **executable** – **[in]** Executable.
- **module_name** – **[in]** Module name. Must be NULL if the symbol has program linkage.
- **symbol_name** – **[in]** Symbol name.
- **agent** – **[in]** Agent associated with the symbol. If the symbol is independent of any agent (for example, a variable with program allocation), this argument is ignored.
- **call_convention** – **[in]** Call convention associated with the symbol. If the symbol does not correspond to an indirect function, this argument is ignored.
- **symbol** – **[out]** Memory location where the HSA runtime stores the symbol handle.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.

- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_INVALID_SYMBOL_NAME** – There is no symbol with a name that matches `symbol_name`.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `symbol_name` is NULL, or `symbol` is NULL.

hsa_status_t HSA_API hsa_executable_get_symbol_by_name (hsa_executable_t executable, const char *symbol_name, const hsa_agent_t *agent, hsa_executable_symbol_t *symbol)

Retrieve the symbol handle corresponding to a given a symbol name.

Parameters

- **executable** – **[in]** Executable.
- **symbol_name** – **[in]** Symbol name. Must be a NUL-terminated character array. The Programmer's Reference Manual describes the standard name mangling scheme.
- **agent** – **[in]** Pointer to the agent for which the symbol with the given name is defined. If the symbol corresponding to the given name has program allocation, `agent` must be NULL.
- **symbol** – **[out]** Memory location where the HSA runtime stores the symbol handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_INVALID_SYMBOL_NAME** – There is no symbol with a name that matches `symbol_name`.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `symbol_name` is NULL, or `symbol` is NULL.

hsa_status_t HSA_API hsa_executable_symbol_get_info (hsa_executable_symbol_t executable_symbol, hsa_executable_symbol_info_t attribute, void *value)

Get the current value of an attribute for a given executable symbol.

Parameters

- **executable_symbol** – **[in]** Executable symbol.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE_SYMBOL** – The executable symbol is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid executable symbol attribute, or `value` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_executable_iterate_symbols (hsa_executable_t executable, hsa_status_t(*callback)(hsa_executable_t exec, hsa_executable_symbol_t symbol, void *data), void *data)

Iterate over the symbols in a executable, and invoke an application-defined callback on every iteration.

Deprecated:

Parameters

- **executable** – **[in]** Executable.
- **callback** – **[in]** Callback to be invoked once per executable symbol. The HSA runtime passes three arguments to the callback: the executable, a symbol, and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and *hsa_executable_iterate_symbols* returns that status value.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API hsa_executable_iterate_agent_symbols (hsa_executable_t executable, hsa_agent_t agent, hsa_status_t(*callback)(hsa_executable_t exec, hsa_agent_t agent, hsa_executable_symbol_t symbol, void *data), void *data)

Iterate over the kernels, indirect functions, and agent allocation variables in an executable for a given agent, and invoke an application- defined callback on every iteration.

Parameters

- **executable** – **[in]** Executable.
- **agent** – **[in]** Agent.
- **callback** – **[in]** Callback to be invoked once per executable symbol. The HSA runtime passes three arguments to the callback: the executable, a symbol, and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and *hsa_executable_iterate_symbols* returns that status value.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API hsa_executable_iterate_program_symbols (hsa_executable_t executable, hsa_status_t(*callback)(hsa_executable_t exec, hsa_executable_symbol_t symbol, void *data), void *data)

Iterate over the program allocation variables in an executable, and invoke an application-defined callback on every iteration.

Parameters

- **executable** – **[in]** Executable.
- **callback** – **[in]** Callback to be invoked once per executable symbol. The HSA runtime passes three arguments to the callback: the executable, a symbol, and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and *hsa_executable_iterate_symbols* returns that status value.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_EXECUTABLE** – The executable is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

struct **hsa_code_object_reader_s**

#include <*hsa.h*> Code object reader handle. A code object reader is used to load a code object from file (when created using *hsa_code_object_reader_create_from_file*), or from memory (if created using *hsa_code_object_reader_create_from_memory*).

struct **hsa_executable_s**

#include <*hsa.h*> Struct containing an opaque handle to an executable, which contains ISA for finalized kernels and indirect functions together with the allocated global or readonly segment variables they reference.

struct **hsa_loaded_code_object_s**

#include <*hsa.h*> Loaded code object handle.

struct **hsa_executable_symbol_s**

#include <*hsa.h*> Executable symbol handle.

The lifetime of an executable object symbol matches that of the executable associated with it. An operation on a symbol whose associated executable has been destroyed results in undefined behavior.

4.4 Finalization extensions

enum **[anonymous]**

Enumeration constants added to *hsa_status_t* by this extension.

Values:

enumerator **HSA_EXT_STATUS_ERROR_INVALID_PROGRAM**

The HSAIL program is invalid.

enumerator **HSA_EXT_STATUS_ERROR_INVALID_MODULE**

The HSAIL module is invalid.

enumerator **HSA_EXT_STATUS_ERROR_INCOMPATIBLE_MODULE**

Machine model or profile of the HSAIL module do not match the machine model or profile of the HSAIL program.

enumerator **HSA_EXT_STATUS_ERROR_MODULE_ALREADY_INCLUDED**

The HSAIL module is already a part of the HSAIL program.

enumerator **HSA_EXT_STATUS_ERROR_SYMBOL_MISMATCH**

Compatibility mismatch between symbol declaration and symbol definition.

enumerator **HSA_EXT_STATUS_ERROR_FINALIZATION_FAILED**

The finalization encountered an error while finalizing a kernel or indirect function.

enumerator **HSA_EXT_STATUS_ERROR_DIRECTIVE_MISMATCH**

Mismatch between a directive in the control directive structure and in the HSAIL kernel.

4.5 Finalization program

enum **hsa_ext_program_info_t**

HSAIL program attributes.

Values:

enumerator **HSA_EXT_PROGRAM_INFO_MACHINE_MODEL**

Machine model specified when the HSAIL program was created. The type of this attribute is *hsa_machine_model_t*.

enumerator **HSA_EXT_PROGRAM_INFO_PROFILE**

Profile specified when the HSAIL program was created. The type of this attribute is *hsa_profile_t*.

enumerator **HSA_EXT_PROGRAM_INFO_DEFAULT_FLOAT_ROUNDING_MODE**

Default float rounding mode specified when the HSAIL program was created. The type of this attribute is *hsa_default_float_rounding_mode_t*.

enum **hsa_ext_finalizer_call_convention_t**

Finalizer-determined call convention.

Values:

enumerator **HSA_EXT_FINALIZER_CALL_CONVENTION_AUTO**

Finalizer-determined call convention.

typedef BrigModule_t **hsa_ext_module_t**

HSAIL (BRIG) module. The HSA Programmer's Reference Manual contains the definition of the BrigModule_t type.

typedef struct *hsa_ext_program_s* **hsa_ext_program_t**

An opaque handle to a HSAIL program, which groups a set of HSAIL modules that collectively define functions and variables used by kernels and indirect functions.

typedef struct *hsa_ext_control_directives_s* **hsa_ext_control_directives_t**

Control directives specify low-level information about the finalization process.

hsa_status_t HSA_API hsa_ext_program_create (hsa_machine_model_t machine_model, hsa_profile_t profile, hsa_default_float_rounding_mode_t default_float_rounding_mode, const char *options, hsa_ext_program_t *program)

Create an empty HSAIL program.

Parameters

- **machine_model** – **[in]** Machine model used in the HSAIL program.
- **profile** – **[in]** Profile used in the HSAIL program.
- **default_float_rounding_mode** – **[in]** Default float rounding mode used in the HSAIL program.
- **options** – **[in]** Vendor-specific options. May be NULL.
- **program** – **[out]** Memory location where the HSA runtime stores the newly created HSAIL program handle.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – There is a failure to allocate resources required for the operation.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `machine_model` is invalid, `profile` is invalid, `default_float_rounding_mode` is invalid, or `program` is NULL.

hsa_status_t HSA_API hsa_ext_program_destroy (hsa_ext_program_t program)

Destroy a HSAIL program.

The HSAIL program handle becomes invalid after it has been destroyed. Code object handles produced by *hsa_ext_program_finalize* are still valid after the HSAIL program has been destroyed, and can be used as intended. Resources allocated outside and associated with the HSAIL program (such as HSAIL modules that are added to the HSAIL program) can be released after the finalization program has been destroyed.

Parameters

program – **[in]** HSAIL program.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

- **HSA_EXT_STATUS_ERROR_INVALID_PROGRAM** – The HSAIL program is invalid.

hsa_status_t HSA_API hsa_ext_program_add_module (hsa_ext_program_t program, hsa_ext_module_t module)

Add a HSAIL module to an existing HSAIL program.

The HSA runtime does not perform a deep copy of the HSAIL module upon addition. Instead, it stores a pointer to the HSAIL module. The ownership of the HSAIL module belongs to the application, which must ensure that `module` is not released before destroying the HSAIL program.

The HSAIL module is successfully added to the HSAIL program if `module` is valid, if all the declarations and definitions for the same symbol are compatible, and if `module` specify machine model and profile that matches the HSAIL program.

Parameters

- **program** – **[in]** HSAIL program.
- **module** – **[in]** HSAIL module. The application can add the same HSAIL module to `program` at most once. The HSAIL module must specify the same machine model and profile as `program`. If the floating-mode rounding mode of `module` is not default, then it should match that of `program`.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – There is a failure to allocate resources required for the operation.
- **HSA_EXT_STATUS_ERROR_INVALID_PROGRAM** – The HSAIL program is invalid.
- **HSA_EXT_STATUS_ERROR_INVALID_MODULE** – The HSAIL module is invalid.
- **HSA_EXT_STATUS_ERROR_INCOMPATIBLE_MODULE** – The machine model of `module` does not match machine model of `program`, or the profile of `module` does not match profile of `program`.
- **HSA_EXT_STATUS_ERROR_MODULE_ALREADY_INCLUDED** – The HSAIL module is already a part of the HSAIL program.
- **HSA_EXT_STATUS_ERROR_SYMBOL_MISMATCH** – Symbol declaration and symbol definition compatibility mismatch. See the symbol compatibility rules in the HSA Programming Reference Manual.

hsa_status_t HSA_API hsa_ext_program_iterate_modules (hsa_ext_program_t program, hsa_status_t(*callback)(hsa_ext_program_t program, hsa_ext_module_t module, void *data), void *data)

Iterate over the HSAIL modules in a program, and invoke an application-defined callback on every iteration.

Parameters

- **program** – **[in]** HSAIL program.
- **callback** – **[in]** Callback to be invoked once per HSAIL module in the program. The HSA runtime passes three arguments to the callback: the program, a HSAIL module, and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and *hsa_ext_program_iterate_modules* returns that status value.

- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_EXT_STATUS_ERROR_INVALID_PROGRAM** – The program is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API hsa_ext_program_get_info (hsa_ext_program_t program, hsa_ext_program_info_t attribute, void *value)

Get the current value of an attribute for a given HSAIL program.

Parameters

- **program** – **[in]** HSAIL program.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behaviour is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_EXT_STATUS_ERROR_INVALID_PROGRAM** – The HSAIL program is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid HSAIL program attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_ext_program_finalize (hsa_ext_program_t program, hsa_isa_t isa, int32_t call_convention, hsa_ext_control_directives_t control_directives, const char *options, hsa_code_object_type_t code_object_type, hsa_code_object_t *code_object)

Finalize an HSAIL program for a given instruction set architecture.

Finalize all of the kernels and indirect functions that belong to the same HSAIL program for a specific instruction set architecture (ISA). The transitive closure of all functions specified by call or scall must be defined. Kernels and indirect functions that are being finalized must be defined. Kernels and indirect functions that are referenced in kernels and indirect functions being finalized may or may not be defined, but must be declared. All the global/readonly segment variables that are referenced in kernels and indirect functions being finalized may or may not be defined, but must be declared.

Parameters

- **program** – **[in]** HSAIL program.
- **isa** – **[in]** Instruction set architecture to finalize for.
- **call_convention** – **[in]** A call convention used in a finalization. Must have a value between *HSA_EXT_FINALIZER_CALL_CONVENTION_AUTO* (inclusive) and the value of the attribute *HSA_ISA_INFO_CALL_CONVENTION_COUNT* in `isa` (not inclusive).
- **control_directives** – **[in]** Low-level control directives that influence the finalization process.
- **options** – **[in]** Vendor-specific options. May be NULL.

- **code_object_type** – **[in]** Type of code object to produce.
- **code_object** – **[out]** Code object generated by the Finalizer, which contains the machine code for the kernels and indirect functions in the HSAIL program. The code object is independent of the HSAIL module that was used to generate it.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – There is a failure to allocate resources required for the operation.
- **HSA_EXT_STATUS_ERROR_INVALID_PROGRAM** – The HSAIL program is invalid.
- **HSA_STATUS_ERROR_INVALID_ISA** – `isa` is invalid.
- **HSA_EXT_STATUS_ERROR_DIRECTIVE_MISMATCH** – The directive in the control directive structure and in the HSAIL kernel mismatch, or if the same directive is used with a different value in one of the functions used by this kernel.
- **HSA_EXT_STATUS_ERROR_FINALIZATION_FAILED** – The Finalizer encountered an error while compiling a kernel or an indirect function.

struct **hsa_ext_program_s**

#include <*hsa_ext_finalize.h*> An opaque handle to a HSAIL program, which groups a set of HSAIL modules that collectively define functions and variables used by kernels and indirect functions.

struct **hsa_ext_control_directives_s**

#include <*hsa_ext_finalize.h*> Control directives specify low-level information about the finalization process.

4.6 Images and samplers

enum **[anonymous]**

Enumeration constants added to *hsa_status_t* by this extension.

Remark

Additions to hsa_status_t

Values:

enumerator **HSA_EXT_STATUS_ERROR_IMAGE_FORMAT_UNSUPPORTED**

Image format is not supported.

enumerator **HSA_EXT_STATUS_ERROR_IMAGE_SIZE_UNSUPPORTED**

Image size is not supported.

enumerator **HSA_EXT_STATUS_ERROR_IMAGE_PITCH_UNSUPPORTED**

Image pitch is not supported or invalid.

enumerator **HSA_EXT_STATUS_ERROR_SAMPLER_DESCRIPTOR_UNSUPPORTED**

Sampler descriptor is not supported or invalid.

enum **[anonymous]**

Enumeration constants added to *hsa_agent_info_t* by this extension.

Remark

Additions to hsa_agent_info_t

Values:

enumerator **HSA_EXT_AGENT_INFO_IMAGE_1D_MAX_ELEMENTS**

Maximum number of elements in 1D images. Must be at least 16384. The type of this attribute is size_t.

enumerator **HSA_EXT_AGENT_INFO_IMAGE_1DA_MAX_ELEMENTS**

Maximum number of elements in 1DA images. Must be at least 16384. The type of this attribute is size_t.

enumerator **HSA_EXT_AGENT_INFO_IMAGE_1DB_MAX_ELEMENTS**

Maximum number of elements in 1DB images. Must be at least 65536. The type of this attribute is size_t.

enumerator **HSA_EXT_AGENT_INFO_IMAGE_2D_MAX_ELEMENTS**

Maximum dimensions (width, height) of 2D images, in image elements. The X and Y maximums must be at least 16384. The type of this attribute is size_t[2].

enumerator **HSA_EXT_AGENT_INFO_IMAGE_2DA_MAX_ELEMENTS**

Maximum dimensions (width, height) of 2DA images, in image elements. The X and Y maximums must be at least 16384. The type of this attribute is size_t[2].

enumerator **HSA_EXT_AGENT_INFO_IMAGE_2DDEPTH_MAX_ELEMENTS**

Maximum dimensions (width, height) of 2DDEPTH images, in image elements. The X and Y maximums must be at least 16384. The type of this attribute is size_t[2].

enumerator **HSA_EXT_AGENT_INFO_IMAGE_2DADEPTH_MAX_ELEMENTS**

Maximum dimensions (width, height) of 2DADEPTH images, in image elements. The X and Y maximums must be at least 16384. The type of this attribute is size_t[2].

enumerator **HSA_EXT_AGENT_INFO_IMAGE_3D_MAX_ELEMENTS**

Maximum dimensions (width, height, depth) of 3D images, in image elements. The maximum along any dimension must be at least 2048. The type of this attribute is size_t[3].

enumerator **HSA_EXT_AGENT_INFO_IMAGE_ARRAY_MAX_LAYERS**

Maximum number of image layers in a image array. Must be at least 2048. The type of this attribute is size_t.

enumerator **HSA_EXT_AGENT_INFO_MAX_IMAGE_RD_HANDLES**

Maximum number of read-only image handles that can be created for an agent at any one time. Must be at least 128. The type of this attribute is size_t.

enumerator **HSA_EXT_AGENT_INFO_MAX_IMAGE_RORW_HANDLES**

Maximum number of write-only and read-write image handles (combined) that can be created for an agent at any one time. Must be at least 64. The type of this attribute is size_t.

enumerator **HSA_EXT_AGENT_INFO_MAX_SAMPLER_HANDLERS**

Maximum number of sampler handlers that can be created for an agent at any one time. Must be at least 16. The type of this attribute is size_t.

enumerator **HSA_EXT_AGENT_INFO_IMAGE_LINEAR_ROW_PITCH_ALIGNMENT**

Image pitch alignment. The agent only supports linear image data layouts with a row pitch that is a multiple of this value. Must be a power of 2. The type of this attribute is size_t.

enum **hsa_ext_image_geometry_t**

Geometry associated with the image. This specifies the number of image dimensions and whether the image is an image array. See the *Image Geometry* section in the *HSA Programming Reference Manual* for definitions on each geometry. The enumeration values match the BRIG type `hsa_ext_brig_image_geometry_t`.

Values:

enumerator **HSA_EXT_IMAGE_GEOMETRY_1D**

One-dimensional image addressed by width coordinate.

enumerator **HSA_EXT_IMAGE_GEOMETRY_2D**

Two-dimensional image addressed by width and height coordinates.

enumerator **HSA_EXT_IMAGE_GEOMETRY_3D**

Three-dimensional image addressed by width, height, and depth coordinates.

enumerator **HSA_EXT_IMAGE_GEOMETRY_1DA**

Array of one-dimensional images with the same size and format. 1D arrays are addressed by width and index coordinate.

enumerator **HSA_EXT_IMAGE_GEOMETRY_2DA**

Array of two-dimensional images with the same size and format. 2D arrays are addressed by width, height, and index coordinates.

enumerator **HSA_EXT_IMAGE_GEOMETRY_1DB**

One-dimensional image addressed by width coordinate. It has specific restrictions compared

to *HSA_EXT_IMAGE_GEOMETRY_1D*. An image with an opaque image data layout will always use a linear image data layout, and one with an explicit image data layout must specify *HSA_EXT_IMAGE_DATA_LAYOUT_LINEAR*.

enumerator **HSA_EXT_IMAGE_GEOMETRY_2DDEPTH**

Two-dimensional depth image addressed by width and height coordinates.

enumerator **HSA_EXT_IMAGE_GEOMETRY_2DADEPTH**

Array of two-dimensional depth images with the same size and format. 2D arrays are addressed by width, height, and index coordinates.

enum **hsa_ext_image_channel_type_t**

Channel type associated with the elements of an image. See the *Channel Type* section in the *HSA Programming Reference Manual* for definitions on each channel type. The enumeration values and definition match the BRIG type `hsa_ext_brig_image_channel_type_t`.

Values:

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_SNORM_INT8**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_SNORM_INT16**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNORM_INT8**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNORM_INT16**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNORM_INT24**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNORM_SHORT_555**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNORM_SHORT_565**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNORM_SHORT_101010**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_SIGNED_INT8**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_SIGNED_INT16**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_SIGNED_INT32**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNSIGNED_INT8**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNSIGNED_INT16**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_UNSIGNED_INT32**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_HALF_FLOAT**

enumerator **HSA_EXT_IMAGE_CHANNEL_TYPE_FLOAT**

enum **hsa_ext_image_channel_order_t**

Channel order associated with the elements of an image. See the *Channel Order* section in the *HSA Programming Reference Manual* for definitions on each channel order. The enumeration values match the BRIG type `hsa_ext_brig_image_channel_order_t`.

Values:

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_A**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_R**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_RX**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_RG**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_RGX**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_RA**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_RGB**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_RGBX**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_RGBA**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_BGRA**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_ARGB**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_ABGR**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_SRGB**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_SRGBX**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_SRGBA**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_SBGRA**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_INTENSITY**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_LUMINANCE**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_DEPTH**

enumerator **HSA_EXT_IMAGE_CHANNEL_ORDER_DEPTH_STENCIL**

enum **hsa_ext_image_capability_t**

Image capability.

Values:

enumerator **HSA_EXT_IMAGE_CAPABILITY_NOT_SUPPORTED**

Images of this geometry, format, and layout are not supported by the agent.

enumerator **HSA_EXT_IMAGE_CAPABILITY_READ_ONLY**

Read-only images of this geometry, format, and layout are supported by the agent.

enumerator **HSA_EXT_IMAGE_CAPABILITY_WRITE_ONLY**

Write-only images of this geometry, format, and layout are supported by the agent.

enumerator **HSA_EXT_IMAGE_CAPABILITY_READ_WRITE**

Read-write images of this geometry, format, and layout are supported by the agent.

enumerator **HSA_EXT_IMAGE_CAPABILITY_READ_MODIFY_WRITE**

Deprecated:

Images of this geometry, format, and layout can be accessed from read-modify-write atomic operations in the agent.

enumerator **HSA_EXT_IMAGE_CAPABILITY_ACCESS_INVARIANT_DATA_LAYOUT**

Images of this geometry, format, and layout are guaranteed to have a consistent data layout regardless of how they are accessed by the associated agent.

enum **hsa_ext_image_data_layout_t**

Image data layout.

An image data layout denotes such aspects of image data layout as tiling and organization of channels in memory. Some image data layouts may only apply to specific image geometries, formats, and access permissions. Different agents may support different image layout identifiers, including vendor specific layouts. Note that an agent may not support the same image data layout for different access permissions to images with the same image geometry, size, and format. If multiple agents support the same image data layout then it is possible to use separate image handles for each agent that references the same image data.

Values:

enumerator **HSA_EXT_IMAGE_DATA_LAYOUT_OPAQUE**

An implementation specific opaque image data layout which can vary depending on the agent, geometry, image format, image size, and access permissions.

enumerator **HSA_EXT_IMAGE_DATA_LAYOUT_LINEAR**

The image data layout is specified by the following rules in ascending byte address order. For a 3D image, 2DA image array, or 1DA image array, the image data is stored as a linear sequence of adjacent 2D image slices, 2D images, or 1D images respectively, spaced according to the slice pitch. Each 2D image is stored as a linear sequence of adjacent image rows, spaced according to the row pitch. Each 1D or 1DB image is stored as a single image row. Each image row is stored as a linear sequence of image elements. Each image element is stored as a linear sequence of image components specified by the left to right channel order definition. Each image component is stored using the memory type specified by the channel type.

The 1DB image geometry always uses the linear image data layout.

enum **hsa_ext_sampler_addressing_mode_t**

Sampler address modes. The sampler address mode describes the processing of out-of-range image coordinates. See the *Addressing Mode* section in the *HSA Programming Reference Manual* for definitions on each address mode. The values match the BRIG type `hsa_ext_brig_sampler_addressing_t`.

Values:

enumerator **HSA_EXT_SAMPLER_ADDRESSING_MODE_UNDEFINED**

Out-of-range coordinates are not handled.

enumerator **HSA_EXT_SAMPLER_ADDRESSING_MODE_CLAMP_TO_EDGE**

Clamp out-of-range coordinates to the image edge.

enumerator **HSA_EXT_SAMPLER_ADDRESSING_MODE_CLAMP_TO_BORDER**

Clamp out-of-range coordinates to the image border color.

enumerator **HSA_EXT_SAMPLER_ADDRESSING_MODE_REPEAT**

Wrap out-of-range coordinates back into the valid coordinate range so the image appears as repeated tiles.

enumerator **HSA_EXT_SAMPLER_ADDRESSING_MODE_MIRRORED_REPEAT**

Mirror out-of-range coordinates back into the valid coordinate range so the image appears as repeated tiles with every other tile a reflection.

enum **hsa_ext_sampler_coordinate_mode_t**

Sampler coordinate normalization modes. See the *Coordinate Normalization Mode* section in the *HSA Programming Reference Manual* for definitions on each coordinate normalization mode. The values match the BRIG type `hsa_ext_brig_sampler_coord_normalization_t`.

Values:

enumerator **HSA_EXT_SAMPLER_COORDINATE_MODE_UNNORMALIZED**

Coordinates are used to directly address an image element.

enumerator **HSA_EXT_SAMPLER_COORDINATE_MODE_NORMALIZED**

Coordinates are scaled by the image dimension size before being used to address an image element.

enum **hsa_ext_sampler_filter_mode_t**

Sampler filter modes. See the *Filter Mode* section in the *HSA Programming Reference Manual* for definitions on each address mode. The enumeration values match the BRIG type `hsa_ext_brig_sampler_filter_t`.

Values:

enumerator **HSA_EXT_SAMPLER_FILTER_MODE_NEAREST**

Filter to the image element nearest (in Manhattan distance) to the specified coordinate.

enumerator **HSA_EXT_SAMPLER_FILTER_MODE_LINEAR**

Filter to the image element calculated by combining the elements in a 2x2 square block or 2x2x2 cube block around the specified coordinate. The elements are combined using linear interpolation.

typedef struct *hsa_amd_image_descriptor_s* **hsa_amd_image_descriptor_t**

Encodes an opaque vendor specific image format. The length of data depends on the underlying format. This structure must not be copied as its true length can not be determined.

typedef struct *hsa_ext_image_s* **hsa_ext_image_t**

Image handle, populated by *hsa_ext_image_create* or *hsa_ext_image_create_with_layout*. Image handles are only unique within an agent, not across agents.

typedef uint32_t **hsa_ext_image_channel_type32_t**

A fixed-size type used to represent *hsa_ext_image_channel_type_t* constants.

typedef uint32_t **hsa_ext_image_channel_order32_t**

A fixed-size type used to represent *hsa_ext_image_channel_order_t* constants.

typedef struct *hsa_ext_image_format_s* **hsa_ext_image_format_t**

Image format.

typedef struct *hsa_ext_image_descriptor_s* **hsa_ext_image_descriptor_t**

Implementation independent image descriptor.

typedef struct *hsa_ext_image_data_info_s* **hsa_ext_image_data_info_t**

Agent specific image size and alignment requirements, populated by *hsa_ext_image_data_get_info* and *hsa_ext_image_data_get_info_with_layout*.

typedef struct *hsa_ext_image_region_s* **hsa_ext_image_region_t**

Image region.

typedef struct *hsa_ext_sampler_s* **hsa_ext_sampler_t**

Sampler handle. Samplers are populated by *hsa_ext_sampler_create* or *hsa_ext_sampler_create_v2*. Sampler handles are only unique within an agent, not across agents.

typedef uint32_t **hsa_ext_sampler_addressing_mode32_t**

A fixed-size type used to represent *hsa_ext_sampler_addressing_mode_t* constants.

typedef uint32_t **hsa_ext_sampler_coordinate_mode32_t**

A fixed-size type used to represent *hsa_ext_sampler_coordinate_mode_t* constants.

typedef uint32_t **hsa_ext_sampler_filter_mode32_t**

A fixed-size type used to represent *hsa_ext_sampler_filter_mode_t* constants.

typedef struct *hsa_ext_sampler_descriptor_s* **hsa_ext_sampler_descriptor_t**

Implementation independent sampler descriptor.

typedef struct *hsa_ext_sampler_descriptor_v2_s* **hsa_ext_sampler_descriptor_v2_t**

Implementation independent sampler descriptor v2 which supports different address modes in X, Y and Z axises.

typedef struct *hsa_ext_images_1_00_pfn_s* **hsa_ext_images_1_00_pfn_t**

The function pointer table for the images v1.00 extension. Can be returned by *hsa_system_get_extension_table* or *hsa_system_get_major_extension_table*.

typedef struct *hsa_ext_images_1_pfn_s* **hsa_ext_images_1_pfn_t**

The function pointer table for the images v1 extension. Can be returned by *hsa_system_get_extension_table* or *hsa_system_get_major_extension_table*.

hsa_status_t HSA_API hsa_amd_image_create (hsa_agent_t agent,
const hsa_ext_image_descriptor_t *image_descriptor,
const hsa_amd_image_descriptor_t *image_layout, const void *image_data,
hsa_access_permission_t access_permission, hsa_ext_image_t *image)

Creates an image from an opaque vendor specific image format. Does not modify data at image_data. Intended initially for accessing interop images.

Parameters

- **agent[in]** – Agent on which to create the image
- **image_descriptor[in]** – **[in]** Vendor specific image format
- **image_data** – **[in]** Pointer to image backing store
- **access_permission** – **[in]** Access permissions for the image object
- **image** – **[out]** Created image object.

Return values

- **HSA_STATUS_SUCCESS** – Image created successfully
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – Bad or mismatched descriptor, null image_data, or mismatched access_permission.

hsa_status_t HSA_API hsa_amd_image_get_info_max_dim (hsa_agent_t agent, hsa_agent_info_t attribute, void *value)

Query image limits.

Parameters

- **agent** – **[in]** A valid agent.
- **attribute** – **[in]** HSA image info attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_QUEUE** – `value` is NULL or `attribute` < HSA_EXT_AGENT_INFO_IMAGE_1D_MAX_ELEMENTS or `attribute` > HSA_EXT_AGENT_INFO_IMAGE_ARRAY_MAX_LAYERS.

hsa_status_t HSA_API hsa_ext_image_get_capability (hsa_agent_t agent, hsa_ext_image_geometry_t geometry, const hsa_ext_image_format_t *image_format, uint32_t *capability_mask)

Retrieve the supported image capabilities for a given combination of agent, geometry, and image format for an image created with an opaque image data layout.

Parameters

- **agent** – **[in]** Agent to be associated with the image handle.
- **geometry** – **[in]** Geometry.
- **image_format** – **[in]** Pointer to an image format. Must not be NULL.
- **capability_mask** – **[out]** Pointer to a memory location where the HSA runtime stores a bit-mask of supported image capability (*hsa_ext_image_capability_t*) values. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `image_format` is NULL, or `capability_mask` is NULL.

hsa_status_t HSA_API hsa_ext_image_get_capability_with_layout (hsa_agent_t agent, hsa_ext_image_geometry_t geometry, const hsa_ext_image_format_t *image_format, hsa_ext_image_data_layout_t image_data_layout, uint32_t *capability_mask)

Retrieve the supported image capabilities for a given combination of agent, geometry, image format, and image layout for an image created with an explicit image data layout.

Parameters

- **agent** – **[in]** Agent to be associated with the image handle.

- **geometry** – **[in]** Geometry.
- **image_format** – **[in]** Pointer to an image format. Must not be NULL.
- **image_data_layout** – **[in]** The image data layout. It is invalid to use *HSA_EXT_IMAGE_DATA_LAYOUT_OPAQUE*; use *hsa_ext_image_get_capability* instead.
- **capability_mask** – **[out]** Pointer to a memory location where the HSA runtime stores a bit-mask of supported image capability (*hsa_ext_image_capability_t*) values. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `image_format` is NULL, `image_data_layout` is *HSA_EXT_IMAGE_DATA_LAYOUT_OPAQUE*, or `capability_mask` is NULL.

hsa_status_t HSA_API hsa_ext_image_data_get_info (hsa_agent_t agent, const hsa_ext_image_descriptor_t *image_descriptor, hsa_access_permission_t access_permission, hsa_ext_image_data_info_t *image_data_info)

Retrieve the image data requirements for a given combination of agent, image descriptor, and access permission for an image created with an opaque image data layout.

The optimal image data size and alignment requirements may vary depending on the image attributes specified in `image_descriptor`, the `access_permission`, and the `agent`. Also, different implementations of the HSA runtime may return different requirements for the same input values.

The implementation must return the same image data requirements for different access permissions with matching image descriptors as long as *hsa_ext_image_get_capability* reports *HSA_EXT_IMAGE_CAPABILITY_ACCESS_INVARIANT_DATA_LAYOUT*. Image descriptors match if they have the same values, with the exception that s-form channel orders match the corresponding non-s-form channel order and vice versa.

Parameters

- **agent** – **[in]** Agent to be associated with the image handle.
- **image_descriptor** – **[in]** Pointer to an image descriptor. Must not be NULL.
- **access_permission** – **[in]** Access permission of the image when accessed by `agent`. The access permission defines how the agent is allowed to access the image and must match the corresponding HSAIL image handle type. The `agent` must support the image format specified in `image_descriptor` for the given `access_permission`.
- **image_data_info** – **[out]** Memory location where the runtime stores the size and alignment requirements. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_EXT_STATUS_ERROR_IMAGE_FORMAT_UNSUPPORTED** – The `agent` does not support the image format specified by `image_descriptor` with the specified `access_permission`.

- **HSA_EXT_STATUS_ERROR_IMAGE_SIZE_UNSUPPORTED** – The agent does not support the image dimensions specified by `image_descriptor` with the specified `access_permission`.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `image_descriptor` is NULL, `access_permission` is not a valid access permission value, or `image_data_info` is NULL.

hsa_status_t HSA_API hsa_ext_image_data_get_info_with_layout (hsa_agent_t agent, const hsa_ext_image_descriptor_t *image_descriptor, hsa_access_permission_t access_permission, hsa_ext_image_data_layout_t image_data_layout, size_t image_data_row_pitch, size_t image_data_slice_pitch, hsa_ext_image_data_info_t *image_data_info)

Retrieve the image data requirements for a given combination of image descriptor, access permission, image data layout, image data row pitch, and image data slice pitch for an image created with an explicit image data layout.

The image data size and alignment requirements may vary depending on the image attributes specified in `image_descriptor`, the `access_permission`, and the image layout. However, different implementations of the HSA runtime will return the same requirements for the same input values.

The implementation must return the same image data requirements for different access permissions with matching image descriptors and matching image layouts as long as *hsa_ext_image_get_capability* reports *HSA_EXT_IMAGE_CAPABILITY_ACCESS_INVARIANT_DATA_LAYOUT*. Image descriptors match if they have the same values, with the exception that s-form channel orders match the corresponding non-s-form channel order and vice versa. Image layouts match if they are the same image data layout and use the same image row and slice pitch values.

Parameters

- **image_descriptor** – **[in]** Pointer to an image descriptor. Must not be NULL.
- **access_permission** – **[in]** Access permission of the image when accessed by an agent. The access permission defines how the agent is allowed to access the image and must match the corresponding HSAIL image handle type.
- **image_data_layout** – **[in]** The image data layout to use. It is invalid to use *HSA_EXT_IMAGE_DATA_LAYOUT_OPAQUE*; use *hsa_ext_image_data_get_info* instead.
- **image_data_row_pitch** – **[in]** The size in bytes for a single row of the image in the image data. If 0 is specified then the default row pitch value is used: image width * image element byte size. The value used must be greater than or equal to the default row pitch, and be a multiple of the image element byte size. For the linear image layout it must also be a multiple of the image linear row pitch alignment for the agents that will access the image data using image instructions.
- **image_data_slice_pitch** – **[in]** The size in bytes of a single slice of a 3D image, or the size in bytes of each image layer in an image array in the image data. If 0 is specified then the default slice pitch value is used: row pitch * height if geometry is *HSA_EXT_IMAGE_GEOMETRY_3D*, *HSA_EXT_IMAGE_GEOMETRY_2DA*, or *HSA_EXT_IMAGE_GEOMETRY_2DADEPTH*; row pitch if geometry is *HSA_EXT_IMAGE_GEOMETRY_1DA*; and 0 otherwise. The value used must be 0 if the default slice pitch is 0, be greater than or equal to the default slice pitch, and be a multiple of the row pitch.
- **image_data_info** – **[out]** Memory location where the runtime stores the size and alignment requirements. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.

- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_EXT_STATUS_ERROR_IMAGE_FORMAT_UNSUPPORTED** – The image format specified by `image_descriptor` is not supported for the `access_permission` and `image_data_layout` specified.
- **HSA_EXT_STATUS_ERROR_IMAGE_SIZE_UNSUPPORTED** – The image dimensions specified by `image_descriptor` are not supported for the `access_permission` and `image_data_layout` specified.
- **HSA_EXT_STATUS_ERROR_IMAGE_PITCH_UNSUPPORTED** – The row and slice pitch specified by `image_data_row_pitch` and `image_data_slice_pitch` are invalid or not supported.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `image_descriptor` is NULL, `image_data_layout` is *HSA_EXT_IMAGE_DATA_LAYOUT_OPAQUE*, or `image_data_info` is NULL.

hsa_status_t HSA_API hsa_ext_image_create (hsa_agent_t agent, const hsa_ext_image_descriptor_t *image_descriptor, const void *image_data, hsa_access_permission_t access_permission, hsa_ext_image_t *image)

Creates an agent specific image handle to an image with an opaque image data layout.

Images with an opaque image data layout created with different access permissions but matching image descriptors and same agent can share the same image data if *HSA_EXT_IMAGE_CAPABILITY_ACCESS_INVARIANT_DATA_LAYOUT* is reported by *hsa_ext_image_get_capability* for the image format specified in the image descriptor. Image descriptors match if they have the same values, with the exception that s-form channel orders match the corresponding non-s-form channel order and vice versa.

If necessary, an application can use image operations (import, export, copy, clear) to prepare the image for the intended use regardless of the access permissions.

Any previous memory contents are preserved upon creation. The application is responsible for ensuring that the lifetime of the image data exceeds that of all the associated images.

support the creation of more image handles with the given `access_permission`).

Parameters

- **agent** – **[in]** agent to be associated with the image handle created.
- **image_descriptor** – **[in]** Pointer to an image descriptor. Must not be NULL.
- **image_data** – **[in]** Image data buffer that must have been allocated according to the size and alignment requirements dictated by *hsa_ext_image_data_get_info*. Must not be NULL.
- **access_permission** – **[in]** Access permission of the image when accessed by agent. The access permission defines how the agent is allowed to access the image using the image handle created and must match the corresponding HSAIL image handle type. The agent must support the image format specified in `image_descriptor` for the given `access_permission`.
- **image** – **[out]** Pointer to a memory location where the HSA runtime stores the newly created image handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_EXT_STATUS_ERROR_IMAGE_FORMAT_UNSUPPORTED** – The agent does not have the capability to support the image format contained in `image_descriptor` using the specified `access_permission`.
- **HSA_EXT_STATUS_ERROR_IMAGE_SIZE_UNSUPPORTED** – The agent does not support the image dimensions specified by `image_descriptor` using the specified `access_permission`.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `image_descriptor` is NULL, `image_data` is NULL, `image_data` does not have a valid alignment, `access_permission` is not a valid access permission value, or `image` is NULL.

hsa_status_t HSA_API hsa_ext_image_create_with_layout (hsa_agent_t agent, const hsa_ext_image_descriptor_t *image_descriptor, const void *image_data, hsa_access_permission_t access_permission, hsa_ext_image_data_layout_t image_data_layout, size_t image_data_row_pitch, size_t image_data_slice_pitch, hsa_ext_image_t *image)

Creates an agent specific image handle to an image with an explicit image data layout.

Images with an explicit image data layout created with different access permissions but matching image descriptors and matching image layout can share the same image data if *HSA_EXT_IMAGE_CAPABILITY_ACCESS_INVARIANT_DATA_LAYOUT* is reported by *hsa_ext_image_get_capability_with_layout* for the image format specified in the image descriptor and specified image data layout. Image descriptors match if they have the same values, with the exception that s-form channel orders match the corresponding non-s-form channel order and vice versa. Image layouts match if they are the same image data layout and use the same image row and slice values.

If necessary, an application can use image operations (import, export, copy, clear) to prepare the image for the intended use regardless of the access permissions.

Any previous memory contents are preserved upon creation. The application is responsible for ensuring that the lifetime of the image data exceeds that of all the associated images.

support the creation of more image handles with the given `access_permission`).

Parameters

- **agent** – **[in]** agent to be associated with the image handle created.
- **image_descriptor** – **[in]** Pointer to an image descriptor. Must not be NULL.
- **image_data** – **[in]** Image data buffer that must have been allocated according to the size and alignment requirements dictated by *hsa_ext_image_data_get_info_with_layout*. Must not be NULL.
- **access_permission** – **[in]** Access permission of the image when accessed by the agent. The access permission defines how the agent is allowed to access the image and must match the corresponding HSAIL image handle type. The agent must support the image format specified in `image_descriptor` for the given `access_permission` and `image_data_layout`.

- **image_data_layout** – **[in]** The image data layout to use for the `image_data`. It is invalid to use *HSA_EXT_IMAGE_DATA_LAYOUT_OPAQUE*; use *hsa_ext_image_create* instead.
- **image_data_row_pitch** – **[in]** The size in bytes for a single row of the image in the image data. If 0 is specified then the default row pitch value is used: image width * image element byte size. The value used must be greater than or equal to the default row pitch, and be a multiple of the image element byte size. For the linear image layout it must also be a multiple of the image linear row pitch alignment for the agents that will access the image data using image instructions.
- **image_data_slice_pitch** – **[in]** The size in bytes of a single slice of a 3D image, or the size in bytes of each image layer in an image array in the image data. If 0 is specified then the default slice pitch value is used: row pitch * height if geometry is *HSA_EXT_IMAGE_GEOMETRY_3D*, *HSA_EXT_IMAGE_GEOMETRY_2DA*, or *HSA_EXT_IMAGE_GEOMETRY_2DADEPTH*; row pitch if geometry is *HSA_EXT_IMAGE_GEOMETRY_1DA*; and 0 otherwise. The value used must be 0 if the default slice pitch is 0, be greater than or equal to the default slice pitch, and be a multiple of the row pitch.
- **image** – **[out]** Pointer to a memory location where the HSA runtime stores the newly created image handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_EXT_STATUS_ERROR_IMAGE_FORMAT_UNSUPPORTED** – The agent does not have the capability to support the image format contained in the image descriptor using the specified `access_permission` and `image_data_layout`.
- **HSA_EXT_STATUS_ERROR_IMAGE_SIZE_UNSUPPORTED** – The agent does not support the image dimensions specified by `image_descriptor` using the specified `access_permission` and `image_data_layout`.
- **HSA_EXT_STATUS_ERROR_IMAGE_PITCH_UNSUPPORTED** – The agent does not support the row and slice pitch specified by `image_data_row_pitch` and `image_data_slice_pitch`, or the values are invalid.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `image_descriptor` is NULL, `image_data` is NULL, `image_data` does not have a valid alignment, `image_data_layout` is *HSA_EXT_IMAGE_DATA_LAYOUT_OPAQUE*, or `image` is NULL.

hsa_status_t HSA_API hsa_ext_image_destroy (hsa_agent_t agent, hsa_ext_image_t image)

Destroy an image handle previously created using *hsa_ext_image_create* or *hsa_ext_image_create_with_layout*.

Destroying the image handle does not free the associated image data, or modify its contents. The application should not destroy an image handle while there are references to it queued for execution or currently being used in a kernel dispatch.

Parameters

- **agent** – **[in]** Agent associated with the image handle.
- **image** – **[in]** Image handle to destroy.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.

hsa_status_t HSA_API hsa_ext_image_copy (hsa_agent_t agent, hsa_ext_image_t src_image, const hsa_dim3_t *src_offset, hsa_ext_image_t dst_image, const hsa_dim3_t *dst_offset, const hsa_dim3_t *range)

Copies a portion of one image (the source) to another image (the destination).

The source and destination image formats should be the same, with the exception that s-form channel orders match the corresponding non-s-form channel order and vice versa. For example, it is allowed to copy a source image with a channel order of HSA_EXT_IMAGE_CHANNEL_ORDER_SRGB to a destination image with a channel order of HSA_EXT_IMAGE_CHANNEL_ORDER_RGB.

The source and destination images do not have to be of the same geometry and appropriate scaling is performed by the HSA runtime. It is possible to copy subregions between any combinations of source and destination geometries, provided that the dimensions of the subregions are the same. For example, it is allowed to copy a rectangular region from a 2D image to a slice of a 3D image.

If the source and destination image data overlap, or the combination of offset and range references an out-out-bounds element in any of the images, the behavior is undefined.

Parameters

- **agent** – **[in]** Agent associated with both the source and destination image handles.
- **src_image** – **[in]** Image handle of source image. The agent associated with the source image handle must be identical to that of the destination image.
- **src_offset** – **[in]** Pointer to the offset within the source image where to copy the data from. Must not be NULL.
- **dst_image** – **[in]** Image handle of destination image.
- **dst_offset** – **[in]** Pointer to the offset within the destination image where to copy the data. Must not be NULL.
- **range** – **[in]** Dimensions of the image portion to be copied. The HSA runtime computes the size of the image data to be copied using this argument. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `src_offset` is NULL, `dst_offset` is NULL, or `range` is NULL.

hsa_status_t HSA_API hsa_ext_image_import (hsa_agent_t agent, const void *src_memory, size_t src_row_pitch, size_t src_slice_pitch, hsa_ext_image_t dst_image, const hsa_ext_image_region_t *image_region)

Import a linearly organized image data from memory directly to an image handle.

This operation updates the image data referenced by the image handle from the source memory. The size of the data imported from memory is implicitly derived from the image region.

It is the application's responsibility to avoid out of bounds memory access.

None of the source memory or destination image data memory can overlap. Overlapping of any of the source and destination image data memory within the import operation produces undefined results.

Parameters

- **agent** – **[in]** Agent associated with the image handle.
- **src_memory** – **[in]** Source memory. Must not be NULL.
- **src_row_pitch** – **[in]** The size in bytes of a single row of the image in the source memory. If the value is smaller than the destination image region width * image element byte size, then region width * image element byte size is used.
- **src_slice_pitch** – **[in]** The size in bytes of a single 2D slice of a 3D image, or the size in bytes of each image layer in an image array in the source memory. If the geometry is *HSA_EXT_IMAGE_GEOMETRY_1DA* and the value is smaller than the value used for `src_row_pitch`, then the value used for `src_row_pitch` is used. If the geometry is *HSA_EXT_IMAGE_GEOMETRY_3D*, *HSA_EXT_IMAGE_GEOMETRY_2DA*, or HSA_EXT_IMAGE_GEOMETRY_2DADEPTH and the value is smaller than the value used for `src_row_pitch` * destination image region height, then the value used for `src_row_pitch` * destination image region height is used. Otherwise, the value is not used.
- **dst_image** – **[in]** Image handle of destination image.
- **image_region** – **[in]** Pointer to the image region to be updated. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `src_memory` is NULL, or `image_region` is NULL.

hsa_status_t HSA_API hsa_ext_image_export (hsa_agent_t agent, hsa_ext_image_t src_image, void *dst_memory, size_t dst_row_pitch, size_t dst_slice_pitch, const hsa_ext_image_region_t *image_region)

Export the image data to linearly organized memory.

The operation updates the destination memory with the image data of `src_image`. The size of the data exported to memory is implicitly derived from the image region.

It is the application's responsibility to avoid out of bounds memory access.

None of the destination memory or source image data memory can overlap. Overlapping of any of the source and destination image data memory within the export operation produces undefined results.

Parameters

- **agent** – **[in]** Agent associated with the image handle.
- **src_image** – **[in]** Image handle of source image.
- **dst_memory** – **[in]** Destination memory. Must not be NULL.
- **dst_row_pitch** – **[in]** The size in bytes of a single row of the image in the destination memory. If the value is smaller than the source image region width * image element byte size, then region width * image element byte size is used.

- **dst_slice_pitch** – **[in]** The size in bytes of a single 2D slice of a 3D image, or the size in bytes of each image in an image array in the destination memory. If the geometry is *HSA_EXT_IMAGE_GEOMETRY_1DA* and the value is smaller than the value used for `dst_row_pitch`, then the value used for `dst_row_pitch` is used. If the geometry is *HSA_EXT_IMAGE_GEOMETRY_3D*, *HSA_EXT_IMAGE_GEOMETRY_2DA*, or HSA_EXT_IMAGE_GEOMETRY_2DADEPTH and the value is smaller than the value used for `dst_row_pitch` * source image region height, then the value used for `dst_row_pitch` * source image region height is used. Otherwise, the value is not used.
- **image_region** – **[in]** Pointer to the image region to be exported. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `dst_memory` is NULL, or `image_region` is NULL.

hsa_status_t HSA_API hsa_ext_image_clear (hsa_agent_t agent, hsa_ext_image_t image, const void *data, const hsa_ext_image_region_t *image_region)

Clear a region of an image so that every image element has the specified value.

Parameters

- **agent** – **[in]** Agent associated with the image handle.
- **image** – **[in]** Image handle for image to be cleared.
- **data** – **[in]** The value to which to set each image element being cleared. It is specified as an array of image component values. The number of array elements must match the number of access components for the image channel order. The type of each array element must match the image access type of the image channel type. When the value is used to set the value of an image element, the conversion method corresponding to the image channel type is used. See the *Channel Order* section and *Channel Type* section in the *HSA Programming Reference Manual* for more information. Must not be NULL.
- **image_region** – **[in]** Pointer to the image region to clear. Must not be NULL. If the region references an out-out-bounds element, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `data` is NULL, or `image_region` is NULL.

hsa_status_t HSA_API hsa_ext_sampler_create (hsa_agent_t agent, const hsa_ext_sampler_descriptor_t *sampler_descriptor, hsa_ext_sampler_t *sampler)

Create an agent specific sampler handle for a given agent independent sampler descriptor and agent.

Parameters

- **agent** – **[in]** Agent to be associated with the sampler handle created.
- **sampler_descriptor** – **[in]** Pointer to a sampler descriptor. Must not be NULL.

- **sampler** – **[out]** Memory location where the HSA runtime stores the newly created sampler handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_EXT_STATUS_ERROR_SAMPLER_DESCRIPTOR_UNSUPPORTED** – The `agent` does not have the capability to support the properties specified by `sampler_descriptor` or it is invalid.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `sampler_descriptor` is NULL, or `sampler` is NULL.

hsa_status_t HSA_API hsa_ext_sampler_create_v2 (hsa_agent_t agent, const hsa_ext_sampler_descriptor_v2_t *sampler_descriptor, hsa_ext_sampler_t *sampler)

Create an agent specific sampler handle for a given agent independent sampler descriptor v2 and agent.

Parameters

- **agent** – **[in]** Agent to be associated with the sampler handle created.
- **sampler_descriptor** – **[in]** v2 Pointer to a sampler descriptor. Must not be NULL.
- **sampler** – **[out]** Memory location where the HSA runtime stores the newly created sampler handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_EXT_STATUS_ERROR_SAMPLER_DESCRIPTOR_UNSUPPORTED** – The `agent` does not have the capability to support the properties specified by `sampler_descriptor` or it is invalid.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `sampler_descriptor` is NULL, or `sampler` is NULL.

hsa_status_t HSA_API hsa_ext_sampler_destroy (hsa_agent_t agent, hsa_ext_sampler_t sampler)

Destroy a sampler handle previously created using *hsa_ext_sampler_create* or *hsa_ext_sampler_create_v2*.

The sampler handle should not be destroyed while there are references to it queued for execution or currently being used in a kernel dispatch.

Parameters

- **agent** – **[in]** Agent associated with the sampler handle.
- **sampler** – **[in]** Sampler handle to destroy.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.

hsa_ext_images_1_00

hsa_ext_images_1

struct **hsa_amd_image_descriptor_s**

#include <hsa_ext_amd.h> Encodes an opaque vendor specific image format. The length of data depends on the underlying format. This structure must not be copied as its true length can not be determined.

struct **hsa_ext_image_s**

#include <hsa_ext_image.h> Image handle, populated by *hsa_ext_image_create* or *hsa_ext_image_create_with_layout*. Image handles are only unique within an agent, not across agents.

struct **hsa_ext_image_format_s**

#include <hsa_ext_image.h> Image format.

struct **hsa_ext_image_descriptor_s**

#include <hsa_ext_image.h> Implementation independent image descriptor.

struct **hsa_ext_image_data_info_s**

#include <hsa_ext_image.h> Agent specific image size and alignment requirements, populated by *hsa_ext_image_data_get_info* and *hsa_ext_image_data_get_info_with_layout*.

struct **hsa_ext_image_region_s**

#include <hsa_ext_image.h> Image region.

struct **hsa_ext_sampler_s**

#include <hsa_ext_image.h> Sampler handle. Samplers are populated by *hsa_ext_sampler_create* or *hsa_ext_sampler_create_v2*. Sampler handles are only unique within an agent, not across agents.

struct **hsa_ext_sampler_descriptor_s**

#include <hsa_ext_image.h> Implementation independent sampler descriptor.

struct **hsa_ext_sampler_descriptor_v2_s**

#include <hsa_ext_image.h> Implementation independent sampler descriptor v2 which supports different address modes in X, Y and Z axises.

struct **hsa_ext_images_1_00_pfn_s**

#include <hsa_ext_image.h> The function pointer table for the images v1.00 extension. Can be returned by *hsa_system_get_extension_table* or *hsa_system_get_major_extension_table*.

struct **hsa_ext_images_1_pfn_s**

#include <hsa_ext_image.h> The function pointer table for the images v1 extension. Can be returned by *hsa_system_get_extension_table* or *hsa_system_get_major_extension_table*.

4.7 Instruction set architecture

enum **hsa_isa_info_t**

Instruction set architecture attributes.

Values:

enumerator **HSA_ISA_INFO_NAME_LENGTH**

The length of the ISA name in bytes, not including the NUL terminator. The type of this attribute is uint32_t.

enumerator **HSA_ISA_INFO_NAME**

Human-readable description. The type of this attribute is character array with the length equal to the value of *HSA_ISA_INFO_NAME_LENGTH* attribute.

enumerator **HSA_ISA_INFO_CALL_CONVENTION_COUNT**

Deprecated:

Number of call conventions supported by the instruction set architecture. Must be greater than zero. The type of this attribute is uint32_t.

enumerator **HSA_ISA_INFO_CALL_CONVENTION_INFO_WAVEFRONT_SIZE**

Deprecated:

Number of work-items in a wavefront for a given call convention. Must be a power of 2 in the range [1,256]. The type of this attribute is uint32_t.

enumerator **HSA_ISA_INFO_CALL_CONVENTION_INFO_WAVEFRONTS_PER_COMPUTE_UNIT**

Deprecated:

Number of wavefronts per compute unit for a given call convention. In practice, other factors (for example, the amount of group memory used by a work-group) may further limit the number of wavefronts per compute unit. The type of this attribute is uint32_t.

enumerator **HSA_ISA_INFO_MACHINE_MODELS**

Machine models supported by the instruction set architecture. The type of this attribute is a bool[2]. If the ISA supports the small machine model, the element at index *HSA_MACHINE_MODEL_SMALL* is true. If the ISA supports the large model, the element at index *HSA_MACHINE_MODEL_LARGE* is true.

enumerator **HSA_ISA_INFO_PROFILES**

Profiles supported by the instruction set architecture. The type of this attribute is a bool[2]. If the ISA supports the base profile, the element at index *HSA_PROFILE_BASE* is true. If the ISA supports the full profile, the element at index *HSA_PROFILE_FULL* is true.

enumerator **HSA_ISA_INFO_DEFAULT_FLOAT_ROUNDING_MODES**

Default floating-point rounding modes supported by the instruction set architecture. The type of this attribute is a bool[3]. The value at a given index is true if the corresponding rounding mode in *hsa_default_float_rounding_mode_t* is supported. At least one default mode has to be supported.

If the default mode is supported, then *HSA_ISA_INFO_BASE_PROFILE_DEFAULT_FLOAT_ROUNDING_MODES* must report that both the zero and the near roundings modes are supported.

enumerator **HSA_ISA_INFO_BASE_PROFILE_DEFAULT_FLOAT_ROUNDING_MODES**

Default floating-point rounding modes supported by the instruction set architecture in the Base profile. The type of this attribute is a bool[3]. The value at a given index is true if the corresponding rounding mode in *hsa_default_float_rounding_mode_t* is supported. The value at index HSA_DEFAULT_FLOAT_ROUNDING_MODE_DEFAULT must be false. At least one of the values at indexes *HSA_DEFAULT_FLOAT_ROUNDING_MODE_ZERO* or HSA_DEFAULT_FLOAT_ROUNDING_MODE_NEAR must be true.

enumerator **HSA_ISA_INFO_FAST_F16_OPERATION**

Flag indicating that the f16 HSAIL operation is at least as fast as the f32 operation in the instruction set architecture. The type of this attribute is bool.

enumerator **HSA_ISA_INFO_WORKGROUP_MAX_DIM**

Maximum number of work-items of each dimension of a work-group. Each maximum must be greater than 0. No maximum can exceed the value of *HSA_ISA_INFO_WORKGROUP_MAX_SIZE*. The type of this attribute is uint16_t[3].

enumerator **HSA_ISA_INFO_WORKGROUP_MAX_SIZE**

Maximum total number of work-items in a work-group. The type of this attribute is uint32_t.

enumerator **HSA_ISA_INFO_GRID_MAX_DIM**

Maximum number of work-items of each dimension of a grid. Each maximum must be greater than 0, and must not be smaller than the corresponding value in *HSA_ISA_INFO_WORKGROUP_MAX_DIM*. No maximum can exceed the value of *HSA_ISA_INFO_GRID_MAX_SIZE*. The type of this attribute is hsa_dim3_t.

enumerator **HSA_ISA_INFO_GRID_MAX_SIZE**

Maximum total number of work-items in a grid. The type of this attribute is uint64_t.

enumerator **HSA_ISA_INFO_FBARRIER_MAX_SIZE**

Maximum number of fbarriers per work-group. Must be at least 32. The type of this attribute is uint32_t.

enum **hsa_fp_type_t**

Floating-point types.

Values:

enumerator **HSA_FP_TYPE_16**

16-bit floating-point type.

enumerator **HSA_FP_TYPE_32**

32-bit floating-point type.

enumerator **HSA_FP_TYPE_64**

64-bit floating-point type.

enum **hsa_flush_mode_t**

Flush to zero modes.

Values:

enumerator **HSA_FLUSH_MODE_FTZ**

Flush to zero.

enumerator **HSA_FLUSH_MODE_NON_FTZ**

Do not flush to zero.

enum **hsa_round_method_t**

Round methods.

Values:

enumerator **HSA_ROUND_METHOD_SINGLE**

Single round method.

enumerator **HSA_ROUND_METHOD_DOUBLE**

Double round method.

enum **hsa_wavefront_info_t**

Wavefront attributes.

Values:

enumerator **HSA_WAVEFRONT_INFO_SIZE**

Number of work-items in the wavefront. Must be a power of 2 in the range [1,256]. The type of this attribute is uint32_t.

typedef struct *hsa_isa_s* **hsa_isa_t**

Instruction set architecture.

typedef struct *hsa_wavefront_s* **hsa_wavefront_t**

Wavefront handle.

hsa_status_t HSA_API hsa_isa_from_name (const char *name, hsa_isa_t *isa)

Retrieve a reference to an instruction set architecture handle out of a symbolic name.

Parameters

- **name** – **[in]** Vendor-specific name associated with a a particular instruction set architecture. `name` must start with the vendor name and a colon (for example, "AMD:"). The rest of the name is vendor-specific. Must be a NUL-terminated string.
- **isa** – **[out]** Memory location where the HSA runtime stores the ISA handle corresponding to the given name. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ISA_NAME** – The given name does not correspond to any instruction set architecture.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `name` is NULL, or `isa` is NULL.

hsa_status_t HSA_API hsa_agent_iterate_isas (hsa_agent_t agent, hsa_status_t(*callback)(hsa_isa_t isa, void *data), void *data)

Iterate over the instruction sets supported by the given agent, and invoke an application-defined callback on every iteration. The iterator is deterministic: if an agent supports several instruction set architectures, they are traversed in the same order in every invocation of this function.

Parameters

- **agent** – **[in]** A valid agent.
- **callback** – **[in]** Callback to be invoked once per instruction set architecture. The HSA runtime passes two arguments to the callback: the ISA and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and that status value is returned.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_isa_get_info (hsa_isa_t isa, hsa_isa_info_t attribute, uint32_t index, void *value)

Get the current value of an attribute for a given instruction set architecture (ISA).

Deprecated:

The concept of call convention has been deprecated. If the application wants to query the value of an attribute for a given instruction set architecture, use *hsa_isa_get_info_alt* instead. If the application wants to query an attribute that is specific to a given combination of ISA and wavefront, use *hsa_wavefront_get_info*.

Parameters

- **isa** – **[in]** A valid instruction set architecture.

- **attribute** – **[in]** Attribute to query.
- **index** – **[in]** Call convention index. Used only for call convention attributes, otherwise ignored. Must have a value between 0 (inclusive) and the value of the attribute *HSA_ISA_INFO_CALL_CONVENTION_COUNT* (not inclusive) in `isa`.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ISA** – The instruction set architecture is invalid.
- **HSA_STATUS_ERROR_INVALID_INDEX** – The index is out of range.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid instruction set architecture attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_isa_get_info_alt (hsa_isa_t isa, hsa_isa_info_t attribute, void *value)

Get the current value of an attribute for a given instruction set architecture (ISA).

Parameters

- **isa** – **[in]** A valid instruction set architecture.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ISA** – The instruction set architecture is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid instruction set architecture attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_isa_get_exception_policies (hsa_isa_t isa, hsa_profile_t profile, uint16_t *mask)

Retrieve the exception policy support for a given combination of instruction set architecture and profile.

Parameters

- **isa** – **[in]** A valid instruction set architecture.
- **profile** – **[in]** Profile.
- **mask** – **[out]** Pointer to a memory location where the HSA runtime stores a mask of *hsa_exception_policy_t* values. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.

- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ISA** – The instruction set architecture is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `profile` is not a valid profile, or `mask` is NULL.

hsa_status_t HSA_API hsa_isa_get_round_method (hsa_isa_t isa, hsa_fp_type_t fp_type, hsa_flush_mode_t flush_mode, hsa_round_method_t *round_method)

Retrieve the round method (single or double) used to implement the floating-point multiply add instruction (mad) for a given combination of instruction set architecture, floating-point type, and flush to zero modifier.

Parameters

- **isa** – **[in]** Instruction set architecture.
- **fp_type** – **[in]** Floating-point type.
- **flush_mode** – **[in]** Flush to zero modifier.
- **round_method** – **[out]** Pointer to a memory location where the HSA runtime stores the round method used by the implementation. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ISA** – The instruction set architecture is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `fp_type` is not a valid floating-point type, or `flush_mode` is not a valid flush to zero modifier, or `round_method` is NULL.

hsa_status_t HSA_API hsa_wavefront_get_info (hsa_wavefront_t wavefront, hsa_wavefront_info_t attribute, void *value)

Get the current value of a wavefront attribute.

Parameters

- **wavefront** – **[in]** A wavefront.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_WAVEFRONT** – The wavefront is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid wavefront attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_isa_iterate_wavefronts (hsa_isa_t isa, hsa_status_t(*callback)(hsa_wavefront_t wavefront, void *data), void *data)

Iterate over the different wavefronts supported by an instruction set architecture, and invoke an application-defined callback on every iteration.

Parameters

- **isa** – **[in]** Instruction set architecture.
- **callback** – **[in]** Callback to be invoked once per wavefront that is supported by the agent. The HSA runtime passes two arguments to the callback: the wavefront handle and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and that value is returned.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ISA** – The instruction set architecture is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_isa_compatible (hsa_isa_t code_object_isa, hsa_isa_t agent_isa, bool *result)

Check if the instruction set architecture of a code object can be executed on an agent associated with another architecture.

Deprecated:

Use *hsa_agent_iterate_isas* to query which instructions set architectures are supported by a given agent.

Parameters

- **code_object_isa** – **[in]** Instruction set architecture associated with a code object.
- **agent_isa** – **[in]** Instruction set architecture associated with an agent.
- **result** – **[out]** Pointer to a memory location where the HSA runtime stores the result of the check. If the two architectures are compatible, the result is true; if they are incompatible, the result is false.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ISA** – `code_object_isa` or `agent_isa` are invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `result` is NULL.

struct **hsa_isa_s**

#include <*hsa.h*> Instruction set architecture.

struct **hsa_wavefront_s**

#include <*hsa.h*> Wavefront handle.

4.8 Memory

enum **hsa_region_segment_t**

Memory segments associated with a region.

Values:

enumerator **HSA_REGION_SEGMENT_GLOBAL**

Global segment. Used to hold data that is shared by all agents.

enumerator **HSA_REGION_SEGMENT_READONLY**

Read-only segment. Used to hold data that remains constant during the execution of a kernel.

enumerator **HSA_REGION_SEGMENT_PRIVATE**

Private segment. Used to hold data that is local to a single work-item.

enumerator **HSA_REGION_SEGMENT_GROUP**

Group segment. Used to hold data that is shared by the work-items of a work-group.

enumerator **HSA_REGION_SEGMENT_KERNARG**

Kernarg segment. Used to store kernel arguments.

enum **hsa_region_global_flag_t**

Global region flags.

Values:

enumerator **HSA_REGION_GLOBAL_FLAG_KERNARG**

The application can use memory in the region to store kernel arguments, and provide the values for the kernarg segment of a kernel dispatch. If this flag is set, then *HSA_REGION_GLOBAL_FLAG_FINE_GRAINED* must be set.

enumerator **HSA_REGION_GLOBAL_FLAG_FINE_GRAINED**

Updates to memory in this region are immediately visible to all the agents under the terms of the HSA memory model. If this flag is set, then *HSA_REGION_GLOBAL_FLAG_COARSE_GRAINED* must not be set.

enumerator **HSA_REGION_GLOBAL_FLAG_COARSE_GRAINED**

Updates to memory in this region can be performed by a single agent at a time. If a different agent in the system is allowed to access the region, the application must explicitely invoke *hsa_memory_assign_agent* in order to transfer ownership to that agent for a particular buffer.

enumerator **HSA_REGION_GLOBAL_FLAG_EXTENDED_SCOPE_FINE_GRAINED**

Updates to memory in this region have extended scope, where the device-scope atomics to this memory type act as system-scope with respect to all variables located in memory regions of this type. Note: On non-compliant systems, the application may still be responsible for performing device-specific actions necessary to achieve system-scope coherence.

enum **hsa_region_info_t**

Attributes of a memory region.

Values:

enumerator **HSA_REGION_INFO_SEGMENT**

Segment where memory in the region can be used. The type of this attribute is *hsa_region_segment_t*.

enumerator **HSA_REGION_INFO_GLOBAL_FLAGS**

Flag mask. The value of this attribute is undefined if the value of *HSA_REGION_INFO_SEGMENT* is not *HSA_REGION_SEGMENT_GLOBAL*. The type of this attribute is uint32_t, a bit-field of *hsa_region_global_flag_t* values.

enumerator **HSA_REGION_INFO_SIZE**

Size of this region, in bytes. The type of this attribute is size_t.

enumerator **HSA_REGION_INFO_ALLOC_MAX_SIZE**

Maximum allocation size in this region, in bytes. Must not exceed the value of *HSA_REGION_INFO_SIZE*. The type of this attribute is size_t.

If the region is in the global or readonly segments, this is the maximum size that the application can pass to *hsa_memory_allocate*.

If the region is in the group segment, this is the maximum size (per work-group) that can be requested for a given kernel dispatch. If the region is in the private segment, this is the maximum size (per work-item) that can be requested for a specific kernel dispatch, and must be at least 256 bytes.

enumerator **HSA_REGION_INFO_ALLOC_MAX_PRIVATE_WORKGROUP_SIZE**

Maximum size (per work-group) of private memory that can be requested for a specific kernel dispatch. Must be at least 65536 bytes. The type of this attribute is uint32_t. The value of this attribute is undefined if the region is not in the private segment.

enumerator **HSA_REGION_INFO_RUNTIME_ALLOC_ALLOWED**

Indicates whether memory in this region can be allocated using *hsa_memory_allocate*. The type of this attribute is bool.

The value of this flag is always false for regions in the group and private segments.

enumerator **HSA_REGION_INFO_RUNTIME_ALLOC_GRANULE**

Allocation granularity of buffers allocated by *hsa_memory_allocate* in this region. The size of a buffer allocated in this region is a multiple of the value of this attribute. The value of this attribute is only defined if *HSA_REGION_INFO_RUNTIME_ALLOC_ALLOWED* is true for this region. The type of this attribute is size_t.

enumerator **HSA_REGION_INFO_RUNTIME_ALLOC_ALIGNMENT**

Alignment of buffers allocated by *hsa_memory_allocate* in this region. The value of this attribute is only defined if *HSA_REGION_INFO_RUNTIME_ALLOC_ALLOWED* is true for this region, and must be a power of 2. The type of this attribute is size_t.

enum **hsa_amd_iommu_version_t**

IOMMU version supported.

Values:

enumerator **HSA_IOMMU_SUPPORT_NONE**

IOMMU not supported

enumerator **HSA_IOMMU_SUPPORT_V2**

IOMMU V2 supported

enum **hsa_amd_agent_info_s**

Agent attributes.

Values:

enumerator **HSA_AMD_AGENT_INFO_CHIP_ID**

Chip identifier. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_CACHELINE_SIZE**

Size of a cacheline in bytes. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_COMPUTE_UNIT_COUNT**

The number of compute unit available in the agent. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_MAX_CLOCK_FREQUENCY**

The maximum clock frequency of the agent in MHz. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_DRIVER_NODE_ID**

Internal driver node identifier. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_MAX_ADDRESS_WATCH_POINTS**

Max number of watch points on memory address ranges to generate exception events when the watched addresses are accessed. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_BDFID**

Agent BDF_ID, named LocationID in thunk. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_MEMORY_WIDTH**

Memory Interface width, the return value type is uint32_t. This attribute is deprecated.

enumerator **HSA_AMD_AGENT_INFO_MEMORY_MAX_FREQUENCY**

Max Memory Clock, the return value type is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_PRODUCT_NAME**

Board name of Agent - populated from MarketingName of Kfd Node The value is an Ascii string of 64 chars.

enumerator **HSA_AMD_AGENT_INFO_MAX_WAVES_PER_CU**

Maximum number of waves possible in a Compute Unit. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_NUM_SIMDS_PER_CU**

Number of SIMD's per compute unit CU The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_NUM_SHADER_ENGINES**

Number of Shader Engines (SE) in Gpu The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_NUM_SHADER_ARRAYS_PER_SE**

Number of Shader Arrays Per Shader Engines in Gpu The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_HDP_FLUSH**

Address of the HDP flush registers. Use of these registers does not conform to the HSA memory model and should be treated with caution. The type of this attribute is hsa_amd_hdp_flush_t.

enumerator **HSA_AMD_AGENT_INFO_DOMAIN**

PCIe domain for the agent. Pairs with HSA_AMD_AGENT_INFO_BDFID to give the full physical location of the Agent. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_COOPERATIVE_QUEUES**

Queries for support of cooperative queues. See *HSA_QUEUE_TYPE_COOPERATIVE*. The type of this attribute is bool.

enumerator **HSA_AMD_AGENT_INFO_UUID**

Queries UUID of an agent. The value is an Ascii string with a maximum of 21 chars including NUL. The string value consists of two parts: header and body. The header identifies device type (GPU, CPU, DSP) while body encodes UUID as a 16 digit hex string

Agents that do not support UUID will return the string "GPU-XX" or "CPU-XX" or "DSP-XX" depending upon their device type *hsa_device_type_t*

enumerator **HSA_AMD_AGENT_INFO_ASIC_REVISION**

Queries for the ASIC revision of an agent. The value is an integer that increments for each revision. This can be used by user-level software to change how it operates, depending on the hardware version. This allows selective workarounds for hardware errata. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_SVM_DIRECT_HOST_ACCESS**

Queries whether or not the host can directly access SVM memory that is physically resident in the agent's local memory. The type of this attribute is bool.

enumerator **HSA_AMD_AGENT_INFO_COOPERATIVE_COMPUTE_UNIT_COUNT**

Some processors support more CUs than can reliably be used in a cooperative dispatch. This queries the count of CUs which are fully enabled for cooperative dispatch. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_MEMORY_AVAIL**

Queries the amount of memory available in bytes accross all global pools owned by the agent. The type of this attribute is uint64_t.

enumerator **HSA_AMD_AGENT_INFO_TIMESTAMP_FREQUENCY**

Timestamp value increase rate, in Hz. The timestamp (clock) frequency is in the range 1-400MHz. The type of this attribute is uint64_t.

enumerator **HSA_AMD_AGENT_INFO_ASIC_FAMILY_ID**

Queries for the ASIC family ID of an agent. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_UCODE_VERSION**

Queries for the Packet Processor(CP Firmware) ucode version of an agent. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_SDMA_UCODE_VERSION**

Queries for the SDMA engine ucode of an agent. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_NUM_SDMA_ENG**

Queries the number of SDMA engines. If HSA_AMD_AGENT_INFO_NUM_SDMA_XGMI_ENG query returns non-zero, this query returns the the number of SDMA engines optimized for host to device bidirectional traffic. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_NUM_SDMA_XGMI_ENG**

Queries the number of additional SDMA engines optimized for D2D xGMI copies. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_IOMMU_SUPPORT**

Queries for version of IOMMU supported by agent. The type of this attribute is hsa_amd_iommu_version_t.

enumerator **HSA_AMD_AGENT_INFO_NUM_XCC**

Queries for number of XCCs within the agent. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_DRIVER_UID**

Queries for driver unique identifier. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_INFO_NEAREST_CPU**

Returns the hsa_agent_t of the nearest CPU agent The type of this attribute is hsa_agent_t.

enumerator **HSA_AMD_AGENT_INFO_MEMORY_PROPERTIES**

Bit-mask indicating memory properties of this agent. A memory property is set if the flag bit is set at that position. User may use the hsa_flag_isset64 macro to verify whether a flag is set. The type of this attribute is uint8_t[8].

enumerator **HSA_AMD_AGENT_INFO_AQL_EXTENSIONS**

Bit-mask indicating AQL Extensions supported by this agent. An AQL extension is set if the flag bit is set at that position. User may use the hsa_flag_isset64 macro to verify whether a flag is set. The type of this attribute is uint8_t[8].

enumerator **HSA_AMD_AGENT_INFO_SCRATCH_LIMIT_MAX**

Maximum allowed value in bytes for scratch limit for this agent. This amount is shared accross all queues created on this agent. The type of this attribute is uint64_t.

enumerator **HSA_AMD_AGENT_INFO_SCRATCH_LIMIT_CURRENT**

Current scratch limit threshold in bytes for this agent. This limit can be modified using the hsa_amd_agent_set_async_scratch_limit call.

- AQL dispatches that require scratch-memory above this threshold will trigger a scratch use-once.
- AQL dispatches using less scratch-memory than this threshold, ROCr will permanently assign the allocated scratch memory to the queue handling the dispatch. This memory can be reclaimed by calling hsa_amd_agent_set_async_scratch_limit with a lower threshold by current value.

The type of this attribute is uint64_t.

enum **hsa_amd_agent_memory_properties_s**

Agent memory properties attributes.

Values:

enumerator **HSA_AMD_MEMORY_PROPERTY_AGENT_IS_APU**

enum **hsa_amd_sdma_engine_id**

SDMA engine IDs unique by single set bit position.

Values:

enumerator **HSA_AMD_SDMA_ENGINE_0**

enumerator **HSA_AMD_SDMA_ENGINE_1**

enumerator **HSA_AMD_SDMA_ENGINE_2**

enumerator **HSA_AMD_SDMA_ENGINE_3**

enumerator **HSA_AMD_SDMA_ENGINE_4**

enumerator **HSA_AMD_SDMA_ENGINE_5**

enumerator **HSA_AMD_SDMA_ENGINE_6**

enumerator **HSA_AMD_SDMA_ENGINE_7**

enumerator **HSA_AMD_SDMA_ENGINE_8**

enumerator **HSA_AMD_SDMA_ENGINE_9**

enumerator **HSA_AMD_SDMA_ENGINE_10**

enumerator **HSA_AMD_SDMA_ENGINE_11**

enumerator **HSA_AMD_SDMA_ENGINE_12**

enumerator **HSA_AMD_SDMA_ENGINE_13**

enumerator **HSA_AMD_SDMA_ENGINE_14**

enumerator **HSA_AMD_SDMA_ENGINE_15**

enum **hsa_amd_region_info_s**

Region attributes.

Values:

enumerator **HSA_AMD_REGION_INFO_HOST_ACCESSIBLE**

Determine if host can access the region. The type of this attribute is bool.

enumerator **HSA_AMD_REGION_INFO_BASE**

Base address of the region in flat address space.

enumerator **HSA_AMD_REGION_INFO_BUS_WIDTH**

Memory Interface width, the return value type is uint32_t. This attribute is deprecated. Use HSA_AMD_AGENT_INFO_MEMORY_WIDTH.

enumerator **HSA_AMD_REGION_INFO_MAX_CLOCK_FREQUENCY**

Max Memory Clock, the return value type is uint32_t. This attribute is deprecated. Use HSA_AMD_AGENT_INFO_MEMORY_MAX_FREQUENCY.

enum **hsa_amd_coherency_type_s**

Coherency attributes of fine grain region.

Values:

enumerator **HSA_AMD_COHERENCY_TYPE_COHERENT**

Coherent region.

enumerator **HSA_AMD_COHERENCY_TYPE_NONCOHERENT**

Non coherent region.

enum **hsa_amd_segment_t**

Memory segments associated with a memory pool.

Values:

enumerator **HSA_AMD_SEGMENT_GLOBAL**

Global segment. Used to hold data that is shared by all agents.

enumerator **HSA_AMD_SEGMENT_READONLY**

Read-only segment. Used to hold data that remains constant during the execution of a kernel.

enumerator **HSA_AMD_SEGMENT_PRIVATE**

Private segment. Used to hold data that is local to a single work-item.

enumerator **HSA_AMD_SEGMENT_GROUP**

Group segment. Used to hold data that is shared by the work-items of a work-group.

enum **hsa_amd_memory_pool_global_flag_s**

Values:

enumerator **HSA_AMD_MEMORY_POOL_GLOBAL_FLAG_KERNARG_INIT**

The application can use allocations in the memory pool to store kernel arguments, and provide the values for the kernarg segment of a kernel dispatch.

enumerator **HSA_AMD_MEMORY_POOL_GLOBAL_FLAG_FINE_GRAINED**

Updates to memory in this pool conform to HSA memory consistency model. If this flag is set, then *HSA_AMD_MEMORY_POOL_GLOBAL_FLAG_COARSE_GRAINED* must not be set.

enumerator **HSA_AMD_MEMORY_POOL_GLOBAL_FLAG_COARSE_GRAINED**

Writes to memory in this pool can be performed by a single agent at a time.

enumerator **HSA_AMD_MEMORY_POOL_GLOBAL_FLAG_EXTENDED_SCOPE_FINE_GRAINED**

Updates to memory in this memory pool have extended scope, acting as system-scope atomics for variables in memory regions of this type. Note: On non-compliant systems, device-specific actions may be required for system-scope coherence.

enum **hsa_amd_memory_pool_location_s**

Values:

enumerator **HSA_AMD_MEMORY_POOL_LOCATION_CPU**

This memory pool resides on the host (CPU)

enumerator **HSA_AMD_MEMORY_POOL_LOCATION_GPU**

This memory pool resides on a GPU

enum **hsa_amd_memory_pool_info_t**

Memory pool features.

Values:

enumerator **HSA_AMD_MEMORY_POOL_INFO_SEGMENT**

Segment where the memory pool resides. The type of this attribute is *hsa_amd_segment_t*.

enumerator **HSA_AMD_MEMORY_POOL_INFO_GLOBAL_FLAGS**

Flag mask. The value of this attribute is undefined if the value of *HSA_AMD_MEMORY_POOL_INFO_SEGMENT* is not *HSA_AMD_SEGMENT_GLOBAL*. The type of this attribute is uint32_t, a bit-field of ::hsa_amd_memory_pool_global_flag_t values.

enumerator **HSA_AMD_MEMORY_POOL_INFO_SIZE**

Size of this pool, in bytes. The type of this attribute is size_t.

enumerator **HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_ALLOWED**

Indicates whether memory in this pool can be allocated using *hsa_amd_memory_pool_allocate*. The type of this attribute is bool.

The value of this flag is always false for memory pools in the group and private segments.

enumerator **HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_GRANULE**

Allocation granularity of buffers allocated by *hsa_amd_memory_pool_allocate* in this memory pool. The size of a buffer allocated in this pool is a multiple of the value of this attribute. While this is the minimum size of allocation allowed, it is recommened to use HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_REC_GRANULE to obtain the recommended allocation granularity size for this pool. The value of this attribute is only defined if *HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_ALLOWED* is true for this pool. The type of this attribute is size_t.

enumerator **HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_ALIGNMENT**

Alignment of buffers allocated by *hsa_amd_memory_pool_allocate* in this pool. The value of this attribute is only defined if *HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_ALLOWED* is true for this pool, and must be a power of 2. The type of this attribute is size_t.

enumerator **HSA_AMD_MEMORY_POOL_INFO_ACCESSIBLE_BY_ALL**

This memory_pool can be made directly accessible by all the agents in the system (*hsa_amd_agent_memory_pool_get_info* does not return *HSA_AMD_MEMORY_POOL_ACCESS_NEVER_ALLOWED* for any agent). The type of this attribute is bool.

enumerator **HSA_AMD_MEMORY_POOL_INFO_ALLOC_MAX_SIZE**

Maximum aggregate allocation size in bytes. The type of this attribute is size_t.

enumerator **HSA_AMD_MEMORY_POOL_INFO_LOCATION**

Location of this memory pool. The type of this attribute is hsa_amd_memory_pool_location_t.

enumerator **HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_REC_GRANULE**

Internal block size for allocations. This would also be the recommended granularity size for allocations as this prevents internal fragmentation. The value of this attribute is only defined if *HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_ALLOWED* is true for this pool. The size of this attribute is size_t.

enum **hsa_amd_memory_pool_flag_s**

Memory pool flag used to specify allocation directives.

Values:

enumerator **HSA_AMD_MEMORY_POOL_STANDARD_FLAG**

Allocates memory that conforms to standard HSA memory consistency model

enumerator **HSA_AMD_MEMORY_POOL_PCIE_FLAG**

Allocates fine grain memory type where memory ordering is per point to point connection. Atomic memory operations on these memory buffers are not guaranteed to be visible at system scope.

enumerator **HSA_AMD_MEMORY_POOL_CONTIGUOUS_FLAG**

Allocates physically contiguous memory

enumerator **HSA_AMD_MEMORY_POOL_EXECUTABLE_FLAG**

Allocates executable memory

enum **hsa_amd_copy_direction_t**

Values:

enumerator **hsaHostToHost**

enumerator **hsaHostToDevice**

enumerator **hsaDeviceToHost**

enumerator **hsaDeviceToDevice**

enum **hsa_amd_memory_pool_access_t**

Type of accesses to a memory pool from a given agent.

Values:

enumerator **HSA_AMD_MEMORY_POOL_ACCESS_NEVER_ALLOWED**

The agent cannot directly access any buffer in the memory pool.

enumerator **HSA_AMD_MEMORY_POOL_ACCESS_ALLOWED_BY_DEFAULT**

The agent can directly access a buffer located in the pool; the application does not need to invoke *hsa_amd_agents_allow_access*.

enumerator **HSA_AMD_MEMORY_POOL_ACCESS_DISALLOWED_BY_DEFAULT**

The agent can directly access a buffer located in the pool, but only if the application has previously requested access to that buffer using *hsa_amd_agents_allow_access*.

enum **hsa_amd_link_info_type_t**

Properties of the relationship between an agent a memory pool.

Values:

enumerator **HSA_AMD_LINK_INFO_TYPE_HYPERTRANSPORT**

Hyper-transport bus type.

enumerator **HSA_AMD_LINK_INFO_TYPE_QPI**

QPI bus type.

enumerator **HSA_AMD_LINK_INFO_TYPE_PCIE**

PCIe bus type.

enumerator **HSA_AMD_LINK_INFO_TYPE_INFINBAND**

Infiniband bus type.

enumerator **HSA_AMD_LINK_INFO_TYPE_XGMI**

xGMI link type.

enum **hsa_amd_agent_memory_pool_info_t**

Properties of the relationship between an agent a memory pool.

Values:

enumerator **HSA_AMD_AGENT_MEMORY_POOL_INFO_ACCESS**

Access to buffers located in the memory pool. The type of this attribute is *hsa_amd_memory_pool_access_t*.

An agent can always directly access buffers currently located in a memory pool that is associated (the memory_pool is one of the values returned by *hsa_amd_agent_iterate_memory_pools* on the agent) with that agent. If the buffer is currently located in a memory pool that is not associated with the agent, and the value returned by this function for the given combination of agent and memory pool is not HSA_AMD_MEMORY_POOL_ACCESS_NEVER_ALLOWED, the application still needs to invoke *hsa_amd_agents_allow_access* in order to gain direct access to the buffer.

If the given agent can directly access buffers the pool, the result is not HSA_AMD_MEMORY_POOL_ACCESS_NEVER_ALLOWED. If the memory pool is associated with the agent, or it is of fined-grained type, the result must not be HSA_AMD_MEMORY_POOL_ACCESS_NEVER_ALLOWED. If the memory pool is not associated with the agent, and does not reside in the global segment, the result must be HSA_AMD_MEMORY_POOL_ACCESS_NEVER_ALLOWED.

enumerator **HSA_AMD_AGENT_MEMORY_POOL_INFO_NUM_LINK_HOPS**

Number of links to hop when accessing the memory pool from the specified agent. The value of this attribute is zero if the memory pool is associated with the agent, or if the access type is HSA_AMD_MEMORY_POOL_ACCESS_NEVER_ALLOWED. The type of this attribute is uint32_t.

enumerator **HSA_AMD_AGENT_MEMORY_POOL_INFO_LINK_INFO**

Details of each link hop when accessing the memory pool starting from the specified agent. The type of this attribute is an array size of HSA_AMD_AGENT_MEMORY_POOL_INFO_NUM_LINK_HOPS with each element containing *hsa_amd_memory_pool_link_info_t*.

enum **hsa_amd_pointer_type_t**

Denotes the type of memory in a pointer info query.

Values:

enumerator **HSA_EXT_POINTER_TYPE_UNKNOWN**

enumerator **HSA_EXT_POINTER_TYPE_HSA**

enumerator **HSA_EXT_POINTER_TYPE_LOCKED**

enumerator **HSA_EXT_POINTER_TYPE_GRAPHICS**

enumerator **HSA_EXT_POINTER_TYPE_IPC**

enumerator **HSA_EXT_POINTER_TYPE_RESERVED_ADDR**

enum **hsa_amd_svm_model_s**

Values:

enumerator **HSA_AMD_SVM_GLOBAL_FLAG_FINE_GRAINED**

Updates to memory with this attribute conform to HSA memory consistency model.

enumerator **HSA_AMD_SVM_GLOBAL_FLAG_COARSE_GRAINED**

Writes to memory with this attribute can be performed by a single agent at a time.

enumerator **HSA_AMD_SVM_GLOBAL_FLAG_INDETERMINATE**

Memory region queried contains subregions with both HSA_AMD_SVM_GLOBAL_FLAG_COARSE_GRAINED and HSA_AMD_SVM_GLOBAL_FLAG_FINE_GRAINED attributes.

This attribute can not be used in hsa_amd_svm_attributes_set. It is a possible return from hsa_amd_svm_attributes_get indicating that the query region contains both coarse and fine grained memory.

enum **hsa_amd_svm_attribute_s**

Values:

enumerator **HSA_AMD_SVM_ATTRIB_GLOBAL_FLAG**

enumerator **HSA_AMD_SVM_ATTRIB_READ_ONLY**

enumerator **HSA_AMD_SVM_ATTRIB_HIVE_LOCAL**

enumerator **HSA_AMD_SVM_ATTRIB_MIGRATION_GRANULARITY**

enumerator **HSA_AMD_SVM_ATTRIB_PREFERRED_LOCATION**

enumerator **HSA_AMD_SVM_ATTRIB_PREFETCH_LOCATION**

enumerator **HSA_AMD_SVM_ATTRIB_READ_MOSTLY**

enumerator **HSA_AMD_SVM_ATTRIB_GPU_EXEC**

enumerator **HSA_AMD_SVM_ATTRIB_AGENT_ACCESSIBLE**

enumerator **HSA_AMD_SVM_ATTRIB_AGENT_ACCESSIBLE_IN_PLACE**

enumerator **HSA_AMD_SVM_ATTRIB_AGENT_NO_ACCESS**

enumerator **HSA_AMD_SVM_ATTRIB_ACCESS_QUERY**

enum **hsa_amd_memory_type_t**

Values:

enumerator **MEMORY_TYPE_NONE**

enumerator **MEMORY_TYPE_PINNED**

typedef struct *hsa_region_s* **hsa_region_t**

A memory region represents a block of virtual memory with certain properties. For example, the HSA runtime represents fine-grained memory in the global segment using a region. A region might be associated with more than one agent.

typedef enum *hsa_amd_agent_info_s* **hsa_amd_agent_info_t**

Agent attributes.

typedef enum *hsa_amd_agent_memory_properties_s* **hsa_amd_agent_memory_properties_t**

Agent memory properties attributes.

typedef enum *hsa_amd_sdma_engine_id* **hsa_amd_sdma_engine_id_t**

SDMA engine IDs unique by single set bit position.

typedef struct *hsa_amd_hdp_flush_s* **hsa_amd_hdp_flush_t**

typedef enum *hsa_amd_region_info_s* **hsa_amd_region_info_t**

Region attributes.

typedef enum *hsa_amd_coherency_type_s* **hsa_amd_coherency_type_t**

Coherency attributes of fine grain region.

typedef struct *hsa_amd_memory_pool_s* **hsa_amd_memory_pool_t**

A memory pool encapsulates physical storage on an agent along with a memory access model.

A memory pool encapsulates a physical partition of an agent's memory system along with a memory access model. Division of a single memory system into separate pools allows querying each partition's access path properties (see *hsa_amd_agent_memory_pool_get_info*). Allocations from a pool are preferentially bound to that pool's physical partition. Binding to the pool's preferential physical partition may not be possible or persistent depending on the system's memory policy and/or state which is beyond the scope of HSA APIs.

For example, a multi-node NUMA memory system may be represented by multiple pool's with each pool providing size and access path information for the partition it represents. Allocations from a pool are preferentially bound to the pool's partition (which in this example is a NUMA node) while following its memory access model. The actual placement may vary or migrate due to the system's NUMA policy and state, which is beyond the scope of HSA APIs.

typedef enum *hsa_amd_memory_pool_global_flag_s* **hsa_amd_memory_pool_global_flag_t**

typedef enum *hsa_amd_memory_pool_location_s* **hsa_amd_memory_pool_location_t**

typedef enum *hsa_amd_memory_pool_flag_s* **hsa_amd_memory_pool_flag_t**

Memory pool flag used to specify allocation directives.

typedef struct *hsa_pitched_ptr_s* **hsa_pitched_ptr_t**

typedef struct *hsa_amd_memory_pool_link_info_s* **hsa_amd_memory_pool_link_info_t**

Link properties when accessing the memory pool from the specified agent.

typedef struct *hsa_amd_pointer_info_s* **hsa_amd_pointer_info_t**

Describes a memory allocation known to ROCr. Within a ROCr major version this structure can only grow.

typedef struct *hsa_amd_ipc_memory_s* **hsa_amd_ipc_memory_t**

256-bit process independent identifier for a ROCr shared memory allocation.

typedef void (***hsa_amd_deallocation_callback_t**)(void *ptr, void *user_data)

Deallocation notifier function type.

typedef enum *hsa_amd_svm_model_s* **hsa_amd_svm_model_t**

typedef enum *hsa_amd_svm_attribute_s* **hsa_amd_svm_attribute_t**

typedef struct *hsa_amd_svm_attribute_pair_s* **hsa_amd_svm_attribute_pair_t**

typedef struct *hsa_amd_vmem_alloc_handle_s* **hsa_amd_vmem_alloc_handle_t**

Struct containing an opaque handle to a memory allocation handle.

typedef struct *hsa_amd_memory_access_desc_s* **hsa_amd_memory_access_desc_t**

hsa_status_t HSA_API hsa_region_get_info (hsa_region_t region, hsa_region_info_t attribute, void *value)

Get the current value of an attribute of a region.

Parameters

- **region** – **[in]** A valid region.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to a application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_REGION** – The region is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid region attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_agent_iterate_regions (hsa_agent_t agent, hsa_status_t(*callback)(hsa_region_t region, void *data), void *data)

Iterate over the memory regions associated with a given agent, and invoke an application-defined callback on every iteration.

Parameters

- **agent** – **[in]** A valid agent.
- **callback** – **[in]** Callback to be invoked once per region that is accessible from the agent. The HSA runtime passes two arguments to the callback, the region and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and *hsa_agent_iterate_regions* returns that status value.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API hsa_memory_allocate (hsa_region_t region, size_t size, void **ptr)

Allocate a block of memory in a given region.

Parameters

- **region** – **[in]** Region where to allocate memory from. The region must have the *HSA_REGION_INFO_RUNTIME_ALLOC_ALLOWED* flag set.
- **size** – **[in]** Allocation size, in bytes. Must not be zero. This value is rounded up to the nearest multiple of *HSA_REGION_INFO_RUNTIME_ALLOC_GRANULE* in `region`.
- **ptr** – **[out]** Pointer to the location where to store the base address of the allocated block. The returned base address is aligned to the value of *HSA_REGION_INFO_RUNTIME_ALLOC_ALIGNMENT* in `region`. If the allocation fails, the returned value is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_REGION** – The region is invalid.
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – The host is not allowed to allocate memory in `region`, or `size` is greater than the value of HSA_REGION_INFO_ALLOC_MAX_SIZE in `region`.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `ptr` is NULL, or `size` is 0.

hsa_status_t HSA_API hsa_memory_free (void *ptr)

Deallocate a block of memory previously allocated using *hsa_memory_allocate*.

Parameters

ptr – **[in]** Pointer to a memory block. If `ptr` does not match a value previously returned by *hsa_memory_allocate*, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

hsa_status_t HSA_API hsa_memory_copy (void *dst, const void *src, size_t size)

Copy a block of memory from the location pointed to by `src` to the memory block pointed to by `dst`.

Parameters

- **dst** – **[out]** Buffer where the content is to be copied. If `dst` is in coarse-grained memory, the copied data is only visible to the agent currently assigned (*hsa_memory_assign_agent*) to `dst`.
- **src** – **[in]** A valid pointer to the source of data to be copied. The source buffer must not overlap with the destination buffer. If the source buffer is in coarse-grained memory then it must be assigned to an agent, from which the data will be retrieved.
- **size** – **[in]** Number of bytes to copy. If `size` is 0, no copy is performed and the function returns success. Copying a number of bytes larger than the size of the buffers pointed by `dst` or `src` results in undefined behavior.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – The source or destination pointers are NULL.

hsa_status_t HSA_API hsa_memory_assign_agent (void *ptr, hsa_agent_t agent, hsa_access_permission_t access)

Change the ownership of a global, coarse-grained buffer.

The contents of a coarse-grained buffer are visible to an agent only after ownership has been explicitely transferred to that agent. Once the operation completes, the previous owner cannot longer access the data in the buffer.

An implementation of the HSA runtime is allowed, but not required, to change the physical location of the buffer when ownership is transferred to a different agent. In general the application must not assume this behavior. The virtual location (address) of the passed buffer is never modified.

Parameters

- **ptr** – **[in]** Base address of a global buffer. The pointer must match an address previously returned by *hsa_memory_allocate*. The size of the buffer affected by the ownership change is identical to the size of that previous allocation. If `ptr` points to a fine-grained global buffer, no operation is performed and the function returns success. If `ptr` does not point to global memory, the behavior is undefined.
- **agent** – **[in]** Agent that becomes the owner of the buffer. The application is responsible for ensuring that `agent` has access to the region that contains the buffer. It is allowed to change ownership to an agent that is already the owner of the buffer, with the same or different access permissions.
- **access** – **[in]** Access permissions requested for the new owner.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `ptr` is NULL, or `access` is not a valid access value.

hsa_status_t HSA_API hsa_memory_register (void *ptr, size_t size)

Register a global, fine-grained buffer.

Registering a buffer serves as an indication to the HSA runtime that the memory might be accessed from a kernel agent other than the host. Registration is a performance hint that allows the HSA runtime implementation to know which buffers will be accessed by some of the kernel agents ahead of time.

Registration is only recommended for buffers in the global segment that have not been allocated using the HSA allocator (*hsa_memory_allocate*), but an OS allocator instead. Registering an OS-allocated buffer in the base profile is equivalent to a no-op.

Registrations should not overlap.

Parameters

- **ptr** – **[in]** A buffer in global, fine-grained memory. If a NULL pointer is passed, no operation is performed. If the buffer has been allocated using *hsa_memory_allocate*, or has already been registered, no operation is performed.
- **size** – **[in]** Requested registration size in bytes. A size of 0 is only allowed if `ptr` is NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `size` is 0 but `ptr` is not NULL.

`hsa_status_t HSA_API hsa_memory_deregister (void *ptr, size_t size)`

Deregister memory previously registered using *hsa_memory_register*.

If the memory interval being deregistered does not match a previous registration (start and end addresses), the behavior is undefined.

Parameters

- `ptr` – **[in]** A pointer to the base of the buffer to be deregistered. If a NULL pointer is passed, no operation is performed.
- `size` – **[in]** Size of the buffer to be deregistered.

Return values

- `HSA_STATUS_SUCCESS` – The function has been executed successfully.
- `HSA_STATUS_ERROR_NOT_INITIALIZED` – The HSA runtime has not been initialized.

`hsa_status_t HSA_API hsa_amd_coherency_get_type (hsa_agent_t agent, hsa_amd_coherency_type_t *type)`

Get the coherency type of the fine grain region of an agent.

Parameters

- `agent` – **[in]** A valid agent.
- `type` – **[out]** Pointer to a memory location where the HSA runtime will store the coherency type of the fine grain region.

Return values

- `HSA_STATUS_SUCCESS` – The function has been executed successfully.
- `HSA_STATUS_ERROR_NOT_INITIALIZED` – The HSA runtime has not been initialized.
- `HSA_STATUS_ERROR_INVALID_AGENT` – The agent is invalid.
- `HSA_STATUS_ERROR_INVALID_ARGUMENT` – `type` is NULL.

`hsa_status_t HSA_API hsa_amd_coherency_set_type (hsa_agent_t agent, hsa_amd_coherency_type_t type)`

Set the coherency type of the fine grain region of an agent. Deprecated. This is supported on KV platforms. For backward compatibility other platforms will spuriously succeed.

Parameters

- `agent` – **[in]** A valid agent.
- `type` – **[in]** The coherency type to be set.

Return values

- `HSA_STATUS_SUCCESS` – The function has been executed successfully.
- `HSA_STATUS_ERROR_NOT_INITIALIZED` – The HSA runtime has not been initialized.
- `HSA_STATUS_ERROR_INVALID_AGENT` – The agent is invalid.
- `HSA_STATUS_ERROR_INVALID_ARGUMENT` – `type` is invalid.

hsa_status_t HSA_API hsa_amd_memory_pool_get_info (hsa_amd_memory_pool_t memory_pool, hsa_amd_memory_pool_info_t attribute, void *value)

Get the current value of an attribute of a memory pool.

Parameters

- **memory_pool** – **[in]** A valid memory pool.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to a application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

HSA_STATUS_SUCCESS – The function has been executed successfully.

hsa_status_t HSA_API hsa_amd_agent_iterate_memory_pools (hsa_agent_t agent, hsa_status_t(*callback)(hsa_amd_memory_pool_t memory_pool, void *data), void *data)

Iterate over the memory pools associated with a given agent, and invoke an application-defined callback on every iteration.

An agent can directly access buffers located in some memory pool, or be enabled to access them by the application (see *hsa_amd_agents_allow_access*), yet that memory pool may not be returned by this function for that given agent.

A memory pool of fine-grained type must be associated only with the host.

Parameters

- **agent** – **[in]** A valid agent.
- **callback** – **[in]** Callback to be invoked on the same thread that called *hsa_amd_agent_iterate_memory_pools*, serially, once per memory pool that is associated with the agent. The HSA runtime passes two arguments to the callback: the memory pool, and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and *hsa_amd_agent_iterate_memory_pools* returns that status value.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API hsa_amd_memory_pool_allocate (hsa_amd_memory_pool_t memory_pool, size_t size, uint32_t flags, void **ptr)

Allocate a block of memory (or buffer) in the specified pool.

Parameters

- **memory_pool** – **[in]** Memory pool where to allocate memory from. The memory pool must have the *HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_ALLOWED* flag set.

- **size** – **[in]** Allocation size, in bytes. Must not be zero. This value is rounded up to the nearest multiple of *HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_GRANULE* in `memory_pool`.
- **flags** – **[in]** A bit-field that is used to specify allocation directives.
- **ptr** – **[out]** Pointer to the location where to store the base virtual address of the allocated block. The returned base address is aligned to the value of *HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_ALIGNMENT* in `memory_pool`. If the allocation fails, the returned value is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – No memory is available.
- **HSA_STATUS_ERROR_INVALID_MEMORY_POOL** – The memory pool is invalid.
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – The host is not allowed to allocate memory in `memory_pool`, or `size` is greater than the value of HSA_AMD_MEMORY_POOL_INFO_ALLOC_MAX_SIZE in `memory_pool`.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `ptr` is NULL, or `size` is 0, or flags is not 0.

hsa_status_t HSA_API hsa_amd_memory_pool_free (void *ptr)

Deallocate a block of memory previously allocated using *hsa_amd_memory_pool_allocate*.

Parameters

ptr – **[in]** Pointer to a memory block. If `ptr` does not match a value previously returned by *hsa_amd_memory_pool_allocate*, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

hsa_status_t HSA_API hsa_amd_memory_async_copy (void *dst, hsa_agent_t dst_agent, const void *src, hsa_agent_t src_agent, size_t size, uint32_t num_dep_signals, const hsa_signal_t *dep_signals, hsa_signal_t completion_signal)

Asynchronously copy a block of memory from the location pointed to by `src` on the `src_agent` to the memory block pointed to by `dst` on the `dst_agent`. Because the DMA engines used may not be in the same coherency domain, the caller must ensure that buffers are system-level coherent. In general this requires the sending device to have released the buffer to system scope prior to executing the copy API and the receiving device must execute a system scope acquire fence prior to use of the destination buffer.

Parameters

- **dst** – **[out]** Buffer where the content is to be copied.
- **dst_agent** – **[in]** Agent associated with the `dst`. The agent must be able to directly access both the source and destination buffers in their current locations. May be zero in which case the runtime will attempt to discover the destination agent. Discovery may have variable and/or high latency.
- **src** – **[in]** A valid pointer to the source of data to be copied. The source buffer must not overlap with the destination buffer, otherwise the copy will succeed but contents of `dst` is undefined.

- **src_agent** – **[in]** Agent associated with the src. The agent must be able to directly access both the source and destination buffers in their current locations. May be zero in which case the runtime will attempt to discover the destination agent. Discovery may have variable and/or high latency.
- **size** – **[in]** Number of bytes to copy. If size is 0, no copy is performed and the function returns success. Copying a number of bytes larger than the size of the buffers pointed by dst or src results in undefined behavior.
- **num_dep_signals** – **[in]** Number of dependent signals. Can be 0.
- **dep_signals** – **[in]** List of signals that must be waited on before the copy operation starts. The copy will start after every signal has been observed with the value 0. The dependent signal should not include completion signal from hsa_amd_memory_async_copy operation to be issued in future as that can result in a deadlock. If num_dep_signals is 0, this argument is ignored.
- **completion_signal** – **[in]** Signal used to indicate completion of the copy operation. When the copy operation is finished, the value of the signal is decremented. The runtime indicates that an error has occurred during the copy operation by setting the value of the completion signal to a negative number. The signal handle must not be 0.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully. The application is responsible for checking for asynchronous error conditions (see the description of completion_signal).
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – An agent is invalid or no discovered agent has access.
- **HSA_STATUS_ERROR_INVALID_SIGNAL** – completion_signal is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – The source or destination pointers are NULL, or the completion signal is 0.

hsa_status_t HSA_API hsa_amd_memory_async_copy_on_engine (void *dst, hsa_agent_t dst_agent, const void *src, hsa_agent_t src_agent, size_t size, uint32_t num_dep_signals, const hsa_signal_t *dep_signals, hsa_signal_t completion_signal, hsa_amd_sdma_engine_id_t engine_id, bool force_copy_on_sdma)

Asynchronously copy a block of memory from the location pointed to by src on the src_agent to the memory block pointed to by dst on the dst_agent on engine_id.

WARNING: Concurrent use of this call with hsa_amd_memory_async_copy can result in resource conflicts as HSA runtime will auto assign engines with the latter call. Approach using both calls concurrently with caution.

All param definitions are identical to hsa_amd_memory_async_copy with the exception of engine_id and force_copy_on_sdma.

All return definitions are identical to hsa_amd_memory_async_copy with the following ammendments:

Parameters

- **-** – **[in]** engine_id Target engine defined by hsa_amd_sdma_engine_id_t. Client should use hsa_amd_memory_copy_engine_status first to get the ID availability.

- **-** – **[in]** force_copy_on_sdma By default, blit kernel copies are used when dst_agent == src_agent. Setting this to true will force the copy over SDMA1.

Return values

HSA_STATUS_ERROR_INVALID_ARGUMENT – The source or destination pointers are NULL, or the completion signal is 0 or engine_id is improperly bounded.

```
hsa_status_t HSA_API hsa_amd_memory_copy_engine_status (hsa_agent_t dst_agent,
hsa_agent_t src_agent, uint32_t *engine_ids_mask)
```

Reports the availability of SDMA copy engines.

Parameters

- **dst_agent** – **[in]** Destination agent of copy status direction.
- **src_agent** – **[in]** Source agent of copy status direction.
- **engine_ids_mask** – **[out]** returns available SDMA engine IDs that can be masked with hsa_amd_sdma_engine_id_t.

Return values

- **HSA_STATUS_SUCCESS** – Agent has available SDMA engines.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Agent does not have available SDMA engines.
- **HSA_STATUS_ERROR_INVALID_AGENT** – dst_agent and src_agent are the same as dst_agent == src_agent is generally used for shader copies.

```
hsa_status_t HSA_API hsa_amd_memory_async_copy_rect (const hsa_pitched_ptr_t *dst,
const hsa_dim3_t *dst_offset, const hsa_pitched_ptr_t *src, const hsa_dim3_t *src_offset,
const hsa_dim3_t *range, hsa_agent_t copy_agent, hsa_amd_copy_direction_t dir,
uint32_t num_dep_signals, const hsa_signal_t *dep_signals,
hsa_signal_t completion_signal)
```

```
hsa_status_t HSA_API hsa_amd_agent_memory_pool_get_info (hsa_agent_t agent,
hsa_amd_memory_pool_t memory_pool, hsa_amd_agent_memory_pool_info_t attribute,
void *value)
```

Get the current value of an attribute of the relationship between an agent and a memory pool.

Parameters

- **agent** – **[in]** Agent.
- **memory_pool** – **[in]** Memory pool.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to a application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

HSA_STATUS_SUCCESS – The function has been executed successfully.

```
hsa_status_t HSA_API hsa_amd_agents_allow_access (uint32_t num_agents,
const hsa_agent_t *agents, const uint32_t *flags, const void *ptr)
```

Enable direct access to a buffer from a given set of agents.

Upon return, only the listed agents and the agent associated with the buffer's memory pool have direct access to the `ptr`.

Any agent that has access to the buffer before and after the call to *hsa_amd_agents_allow_access* will also have access while *hsa_amd_agents_allow_access* is in progress.

The caller is responsible for ensuring that each agent in the list must be able to access the memory pool containing `ptr` (using *hsa_amd_agent_memory_pool_get_info* with *HSA_AMD_AGENT_MEMORY_POOL_INFO_ACCESS* attribute), otherwise error code is returned.

Parameters

- **num_agents** – **[in]** Size of `agents`.
- **agents** – **[in]** List of agents. If `num_agents` is 0, this argument is ignored.
- **flags** – **[in]** A list of bit-field that is used to specify access information in a per-agent basis. This is currently reserved and must be NULL.
- **ptr** – **[in]** A buffer previously allocated using *hsa_amd_memory_pool_allocate*.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `num_agents` is 0, or `agents` is NULL, `flags` is not NULL, or attempting to enable access to agent(s) because `ptr` is allocated from an inaccessible pool.

hsa_status_t HSA_API hsa_amd_memory_pool_can_migrate (hsa_amd_memory_pool_t src_memory_pool, hsa_amd_memory_pool_t dst_memory_pool, bool *result)

Query if buffers currently located in some memory pool can be relocated to a destination memory pool.

If the returned value is non-zero, a migration of a buffer to `dst_memory_pool` using *hsa_amd_memory_migrate* may nevertheless fail due to resource limitations.

Parameters

- **src_memory_pool** – **[in]** Source memory pool.
- **dst_memory_pool** – **[in]** Destination memory pool.
- **result** – **[out]** Pointer to a memory location where the result of the query is stored. Must not be NULL. If buffers currently located in `src_memory_pool` can be relocated to `dst_memory_pool`, the result is true.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_MEMORY_POOL** – One of the memory pools is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `result` is NULL.

hsa_status_t HSA_API hsa_amd_memory_migrate (const void *ptr, hsa_amd_memory_pool_t memory_pool, uint32_t flags)

Relocate a buffer to a new memory pool.

When a buffer is migrated, its virtual address remains the same but its physical contents are moved to the indicated memory pool.

After migration, only the agent associated with the destination pool will have access.

The caller is also responsible for ensuring that the allocation in the source memory pool where the buffer is currently located can be migrated to the specified destination memory pool (using *hsa_amd_memory_pool_can_migrate* returns a value of true for the source and destination memory pools), otherwise behavior is undefined.

The caller must ensure that the buffer is not accessed while it is migrated.

Parameters

- **ptr** – **[in]** Buffer to be relocated. The buffer must have been released to system prior to call this API. The buffer will be released to system upon completion.
- **memory_pool** – **[in]** Memory pool where to place the buffer.
- **flags** – **[in]** A bit-field that is used to specify migration information. Must be zero.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_MEMORY_POOL** – The destination memory pool is invalid.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – There is a failure in allocating the necessary resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `flags` is not 0.

hsa_status_t HSA_API hsa_amd_memory_lock (void *host_ptr, size_t size, hsa_agent_t *agents, int num_agent, void **agent_ptr)

Pin a host pointer allocated by C/C++ or OS allocator (i.e. ordinary system DRAM) and return a new pointer accessible by the `agents`. If the `host_ptr` overlaps with previously locked memory, then the overlap area is kept locked (i.e multiple mappings are permitted). In this case, the same input `host_ptr` may give different locked `agent_ptr` and when it does, they are not necessarily coherent (i.e. accessing either `agent_ptr` is not equivalent). Accesses to `agent_ptr` are coarse grained.

Parameters

- **host_ptr** – **[in]** A buffer allocated by C/C++ or OS allocator.
- **size** – **[in]** The size to be locked.
- **agents** – **[in]** Array of agent handle to gain access to the `host_ptr`. If this parameter is NULL and the `num_agent` is 0, all agents in the platform will gain access to the `host_ptr`.
- **agent_ptr** – **[out]** Pointer to the location where to store the new address.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – There is a failure in allocating the necessary resources.
- **HSA_STATUS_ERROR_INVALID_AGENT** – One or more agent in `agents` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `size` is 0 or `host_ptr` or `agent_ptr` is NULL or `agents` not NULL but `num_agent` is 0 or `agents` is NULL but `num_agent` is not 0.

hsa_status_t HSA_API hsa_amd_memory_lock_to_pool (void *host_ptr, size_t size, hsa_agent_t *agents, int num_agent, hsa_amd_memory_pool_t pool, uint32_t flags, void **agent_ptr)

Pin a host pointer allocated by C/C++ or OS allocator (i.e. ordinary system DRAM) and return a new pointer accessible by the `agents`. If the `host_ptr` overlaps with previously locked memory, then the overlap area is kept locked (i.e. multiple mappings are permitted). In this case, the same input `host_ptr` may give different locked `agent_ptr` and when it does, they are not necessarily coherent (i.e. accessing either `agent_ptr` is not equivalent). Acesses to the memory via `agent_ptr` have the same access properties as memory allocated from `pool` as determined by *hsa_amd_memory_pool_get_info* and *hsa_amd_agent_memory_pool_get_info* (ex. coarse/fine grain, platform atomic support, link info). Physical composition and placement of the memory (ex. page size, NUMA binding) is not changed.

Parameters

- **host_ptr** – **[in]** A buffer allocated by C/C++ or OS allocator.
- **size** – **[in]** The size to be locked.
- **agents** – **[in]** Array of agent handle to gain access to the `host_ptr`. If this parameter is NULL and the `num_agent` is 0, all agents in the platform will gain access to the `host_ptr`.
- **pool** – **[in]** Global memory pool owned by a CPU agent.
- **flags** – **[in]** A bit-field that is used to specify allocation directives. Reserved parameter, must be 0.
- **agent_ptr** – **[out]** Pointer to the location where to store the new address.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – There is a failure in allocating the necessary resources.
- **HSA_STATUS_ERROR_INVALID_AGENT** – One or more agent in `agents` is invalid or can not access `pool`.
- **HSA_STATUS_ERROR_INVALID_MEMORY_POOL** – `pool` is invalid or not owned by a CPU agent.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `size` is 0 or `host_ptr` or `agent_ptr` is NULL or `agents` not NULL but `num_agent` is 0 or `agents` is NULL but `num_agent` is not 0 or flags is not 0.

hsa_status_t HSA_API hsa_amd_memory_unlock (void *host_ptr)

Unpin the host pointer previously pinned via *hsa_amd_memory_lock* or *hsa_amd_memory_lock_to_pool*.

The behavior is undefined if the host pointer being unpinned does not match previous pinned address or if the host pointer was already deallocated.

Parameters

host_ptr – **[in]** A buffer allocated by C/C++ or OS allocator that was pinned previously via *hsa_amd_memory_lock* or *hsa_amd_memory_lock_to_pool*.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

hsa_status_t HSA_API hsa_amd_memory_fill (void *ptr, uint32_t value, size_t count)

Sets the first `count` of uint32_t of the block of memory pointed by `ptr` to the specified `value`.

Parameters

- **ptr** – **[in]** Pointer to the block of memory to fill.
- **value** – **[in]** Value to be set.
- **count** – **[in]** Number of uint32_t element to be set to the value.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `ptr` is NULL or not 4 bytes aligned
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – if the given memory region was not allocated with HSA runtime APIs.

hsa_status_t HSA_API hsa_amd_interop_map_buffer (uint32_t num_agents, hsa_agent_t *agents, int interop_handle, uint32_t flags, size_t *size, void **ptr, size_t *metadata_size, const void **metadata)

Maps an interop object into the HSA flat address space and establishes memory residency. The metadata pointer is valid during the lifetime of the map (until hsa_amd_interop_unmap_buffer is called). Multiple calls to hsa_amd_interop_map_buffer with the same interop_handle result in multiple mappings with potentially different addresses and different metadata pointers. Concurrent operations on these addresses are not coherent. Memory must be fenced to system scope to ensure consistency, between mappings and with any views of this buffer in the originating software stack.

Parameters

- **num_agents** – **[in]** Number of agents which require access to the memory
- **agents** – **[in]** List of accessing agents.
- **interop_handle** – **[in]** Handle of interop buffer (dmabuf handle in Linux)
- **flags** – **[in]** Reserved, must be 0
- **size** – **[out]** Size in bytes of the mapped object
- **ptr** – **[out]** Base address of the mapped object
- **metadata_size** – **[out]** Size of metadata in bytes, may be NULL
- **metadata** – **[out]** Pointer to metadata, may be NULL

Return values

- **HSA_STATUS_SUCCESS** – if successfully mapped
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – all other errors

hsa_status_t HSA_API hsa_amd_interop_unmap_buffer (void *ptr)

Removes a previously mapped interop object from HSA's flat address space. Ends lifetime for the mapping's associated metadata pointer.

hsa_status_t HSA_API hsa_amd_pointer_info (const void *ptr, hsa_amd_pointer_info_t *info, void *(*alloc)(size_t), uint32_t *num_agents_accessible, hsa_agent_t **accessible)

Retrieves information about the allocation referenced by the given pointer. Optionally returns the number and list of agents which can directly access the allocation. In case this virtual address is unknown, the pointer type returned will be HSA_EXT_POINTER_TYPE_UNKNOWN and the only fields that are valid after hsa_amd_pointer_info returns are size and type.

Parameters

- **ptr** – **[in]** Pointer which references the allocation to retrieve info for.
- **info** – **[inout]** Pointer to structure to be filled with allocation info. Data member size must be set to the size of the structure prior to calling hsa_amd_pointer_info. On return size will be set to the size of the pointer info structure supported by the runtime, if smaller. Members beyond the returned value of size will not be updated by the API. Must not be NULL.
- **alloc** – **[in]** Function pointer to an allocator used to allocate the `accessible` array. If NULL `accessible` will not be returned.
- **num_agents_accessible** – **[out]** Recieves the count of agents in `accessible`. If NULL `accessible` will not be returned.
- **accessible** – **[out]** Recieves a pointer to the array, allocated by `alloc`, holding the list of agents which may directly access the allocation. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – Info retrieved successfully
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – NULL in `ptr` or `info`.

hsa_status_t HSA_API hsa_amd_pointer_info_set_userdata (const void *ptr, void *userdata)

Associates an arbitrary pointer with an allocation known to ROCr. The pointer can be fetched by hsa_amd_pointer_info in the userData field.

Parameters

- **ptr** – **[in]** Pointer to the first byte of an allocation known to ROCr with which to associate `userdata`.
- **userdata** – **[in]** Abitrary pointer to associate with the allocation.

Return values

- **HSA_STATUS_SUCCESS** – `userdata` successfully stored.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `ptr` is not known to ROCr.

hsa_status_t HSA_API hsa_amd_ipc_memory_create (void *ptr, size_t len, hsa_amd_ipc_memory_t *handle)

Prepares an allocation for interprocess sharing and creates a handle of type hsa_amd_ipc_memory_t uniquely identifying the allocation. A handle is valid while the allocation it references remains accessible in any

process. In general applications should confirm that a shared memory region has been attached (via hsa_amd_ipc_memory_attach) in the remote process prior to releasing that memory in the local process. Repeated calls for the same allocation may, but are not required to, return unique handles. The allocation needs to be on memory on an agent of type HSA_DEVICE_TYPE_GPU.

Parameters

- **ptr** – **[in]** Pointer to device memory allocated via ROCr APIs to prepare for sharing.
- **len** – **[in]** Length in bytes of the allocation to share.
- **handle** – **[out]** Process independent identifier referencing the shared allocation.

Return values

- **HSA_STATUS_SUCCESS** – allocation is prepared for interprocess sharing.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – ptr does not point to the first byte of an allocation made through ROCr, or len is not the full length of the allocation or handle is NULL.

hsa_status_t HSA_API hsa_amd_ipc_memory_attach (const hsa_amd_ipc_memory_t *handle, size_t len, uint32_t num_agents, const hsa_agent_t *mapping_agents, void **mapped_ptr)

Imports shared memory into the local process and makes it accessible by the given agents. If a shared memory handle is attached multiple times in a process each attach may return a different address. Each returned address is refcounted and requires a matching number of calls to hsa_amd_ipc_memory_detach to release the shared memory mapping.

Parameters

- **handle** – **[in]** Pointer to the identifier for the shared memory.
- **len** – **[in]** Length of the shared memory to import. Reserved. Must be the full length of the shared allocation in this version.
- **num_agents** – **[in]** Count of agents in mapping_agents. May be zero if all agents are to be allowed access.
- **mapping_agents** – **[in]** List of agents to access the shared memory. Ignored if num_agents is zero.
- **mapped_ptr** – **[out]** Recieves a process local pointer to the shared memory.

Return values

- **HSA_STATUS_SUCCESS** – if memory is successfully imported.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – handle is not valid, len is incorrect, mapped_ptr is NULL, or some agent for which access was requested can not access the shared memory.

hsa_status_t HSA_API hsa_amd_ipc_memory_detach (void *mapped_ptr)

Decrements the reference count for the shared memory mapping and releases access to shared memory imported with hsa_amd_ipc_memory_attach.

Parameters

mapped_ptr – **[in]** Pointer to the first byte of a shared allocation imported with hsa_amd_ipc_memory_attach.

Return values

- **HSA_STATUS_SUCCESS** – if mapped_ptr was imported with hsa_amd_ipc_memory_attach.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – mapped_ptr was not imported with hsa_amd_ipc_memory_attach.

hsa_status_t HSA_API hsa_amd_register_deallocation_callback (void *ptr, hsa_amd_deallocation_callback_t callback, void *user_data)

Registers a deallocation notifier monitoring for release of agent accessible address ptr. If successful, callback will be invoked when ptr is removed from accessibility from all agents.

Notification callbacks are automatically deregistered when they are invoked.

Note: The current version supports notifications of address release originating from *hsa_amd_memory_pool_free*. Support for other address release APIs will follow.

Parameters

- **ptr** – **[in]** Agent accessible address to monitor for deallocation. Passed to callback.
- **callback** – **[in]** Notifier to be invoked when ptr is released from agent accessibility.
- **user_data** – **[in]** User provided value passed to callback. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The notifier registered successfully
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – ptr does not refer to a valid agent accessible address.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – callback is NULL or ptr is NULL.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources

hsa_status_t HSA_API hsa_amd_deregister_deallocation_callback (void *ptr, hsa_amd_deallocation_callback_t callback)

Removes a deallocation notifier previously registered with *hsa_amd_register_deallocation_callback*. Arguments must be identical to those given in *hsa_amd_register_deallocation_callback*.

Parameters

- **ptr** – **[in]** Agent accessible address which was monitored for deallocation.
- **callback** – **[in]** Notifier to be removed.

Return values

- **HSA_STATUS_SUCCESS** – The notifier has been removed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – The given notifier was not registered.

hsa_status_t **hsa_amd_svm_attributes_set**(void *ptr, size_t size, *hsa_amd_svm_attribute_pair_t* *attribute_list, size_t attribute_count)

Sets SVM memory attributes.

If HSA_AMD_SYSTEM_INFO_SVM_ACCESSIBLE_BY_DEFAULT returns false then enabling access to an Agent via this API (setting HSA_AMD_SVM_ATTRIB_AGENT_ACCESSIBLE or HSA_AMD_SVM_ATTRIB_AGENT_ACCESSIBLE_IN_PLACE) is required prior to SVM memory access by that Agent.

Attributes HSA_AMD_SVM_ATTRIB_ACCESS_QUERY and HSA_AMD_SVM_ATTRIB_PREFETCH_LOCATION may not be used with this API.

Parameters

- **ptr** – **[in]** Will be aligned down to nearest page boundary.
- **size** – **[in]** Will be aligned up to nearest page boundary.
- **attribute_list** – **[in]** List of attributes to set for the address range.
- **attribute_count** – **[in]** Length of `attribute_list`.

hsa_status_t **hsa_amd_svm_attributes_get**(void *ptr, size_t size, *hsa_amd_svm_attribute_pair_t* *attribute_list, size_t attribute_count)

Gets SVM memory attributes.

Attributes HSA_AMD_SVM_ATTRIB_AGENT_ACCESSIBLE, HSA_AMD_SVM_ATTRIB_AGENT_ACCESSIBLE_IN_PLA and HSA_AMD_SVM_ATTRIB_PREFETCH_LOCATION may not be used with this API.

Note that attribute HSA_AMD_SVM_ATTRIB_ACCESS_QUERY takes as input an hsa_agent_t and returns the current access type through its attribute field.

Parameters

- **ptr** – **[in]** Will be aligned down to nearest page boundary.
- **size** – **[in]** Will be aligned up to nearest page boundary.
- **attribute_list** – **[in]** List of attributes to set for the address range.
- **attribute_count** – **[in]** Length of `attribute_list`.

hsa_status_t **hsa_amd_svm_prefetch_async**(void *ptr, size_t size, *hsa_agent_t* agent, uint32_t num_dep_signals, const *hsa_signal_t* *dep_signals, *hsa_signal_t* completion_signal)

Asynchronously migrates memory to an agent.

Schedules memory migration to `agent` when `dep_signals` have been observed equal to zero. `completion_signal` will decrement when the migration is complete.

Parameters

- **ptr** – **[in]** Will be aligned down to nearest page boundary.
- **size** – **[in]** Will be aligned up to nearest page boundary.
- **agent** – **[in]** Agent to migrate to.
- **num_dep_signals** – **[in]** Number of dependent signals. Can be 0.
- **dep_signals** – **[in]** List of signals that must be waited on before the migration operation starts. The migration will start after every signal has been observed with the value 0. If `num_dep_signals` is 0, this argument is ignored.

- **completion_signal** – **[in]** Signal used to indicate completion of the migration operation. When the migration operation is finished, the value of the signal is decremented. The runtime indicates that an error has occurred during the copy operation by setting the value of the completion signal to a negative number. If no completion signal is required this handle may be null.

hsa_status_t **hsa_amd_portable_export_dmabuf**(const void *ptr, size_t size, int *dmabuf, uint64_t *offset)

Obtains an OS specific, vendor neutral, handle to a memory allocation.

Obtains an OS specific handle to GPU agent memory. The memory must be part of a single allocation from an hsa_amd_memory_pool_t exposed by a GPU Agent. The handle may be used with other APIs (e.g. Vulkan) to obtain shared access to the allocation.

Shared access to the memory is not guaranteed to be fine grain coherent even if the allocation exported is from a fine grain pool. The shared memory consistency model will be no stronger than the model exported from, consult the importing API to determine the final consistency model.

The allocation's memory remains valid as long as the handle and any mapping of the handle remains valid. When the handle and all mappings are closed the backing memory will be released for reuse.

Parameters

- **ptr** – **[in]** Pointer to the allocation being exported.
- **size** – **[in]** Size in bytes to export following `ptr`. The entire range being exported must be contained within a single allocation.
- **dmabuf** – **[out]** Pointer to a dma-buf file descriptor holding a reference to the allocation. Contents will not be altered in the event of failure.
- **offset** – **[out]** Offset in bytes into the memory referenced by the dma-buf object at which `ptr` resides. Contents will not be altered in the event of failure.

Return values

- **HSA_STATUS_SUCCESS** – Export completed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – One or more arguments is NULL.
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – The address range described by `ptr` and `size` are not contained within a single allocation.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The allocation described by `ptr` and `size` was allocated on a device which can not export memory.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The return file descriptor, `dmabuf`, could not be created.

hsa_status_t **hsa_amd_portable_close_dmabuf**(int dmabuf)

Closes an OS specific, vendor neutral, handle to a memory allocation.

Closes an OS specific handle to GPU agent memory.

Applications should close a handle after imports are complete. The handle is not required to remain open for the lifetime of imported mappings. The referenced allocation will remain valid until all handles and mappings are closed.

Parameters

dmabuf – **[in]** Handle to be closed.

Return values

- **HSA_STATUS_SUCCESS** – Handle closed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_RESOURCE_FREE** – A generic error was encountered when closing the handle. The handle may have been closed already or an async IO error may have occured.

hsa_status_t **hsa_amd_vmem_address_reserve**(void **va, size_t size, uint64_t address, uint64_t flags)

Allocate a reserved address range.

Reserve a virtual address range. The size must be a multiple of the system page size. If it is not possible to allocate the address specified by `address`, then `va` will be a different address range. Address range should be released by calling hsa_amd_vmem_address_free.

Note that this API will be deprecated in a future release and replaced by hsa_amd_vmem_address_reserve_align

Parameters

- **va** – **[out]** virtual address allocated
- **size** – **[in]** of address range requested
- **address** – **[in]** requested
- **flags** – **[in]** currently unsupported

Return values

- **HSA_STATUS_SUCCESS** – Address range allocated successfully
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Insufficient resources to allocate an address range of this size.

hsa_status_t **hsa_amd_vmem_address_reserve_align**(void **va, size_t size, uint64_t address, uint64_t alignment, uint64_t flags)

Allocate a reserved address range.

Reserve a virtual address range. The size must be a multiple of the system page size. If it is not possible to allocate the address specified by `address`, then `va` will be a different address range. Address range should be released by calling hsa_amd_vmem_address_free.

Parameters

- **va** – **[out]** virtual address allocated
- **size** – **[in]** of address range requested
- **address** – **[in]** requested
- **alignment** – **[in]** requested. 0 for default. Must be >= page-size and a power of 2
- **flags** – **[in]** currently unsupported

Return values

- **HSA_STATUS_SUCCESS** – Address range allocated successfully
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Insufficient resources to allocate an address range of this size.

hsa_status_t **hsa_amd_vmem_address_free**(void *va, size_t size)

Free a reserved address range.

Free a previously allocated address range. The size must match the size of a previously allocated address range.

Parameters

- **va** – **[out]** virtual address to be freed
- **size** – **[in]** of address range

Return values

- **HSA_STATUS_SUCCESS** – Address range released successfully
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – Invalid va specified
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – Invalid size specified
- **HSA_STATUS_ERROR_RESOURCE_FREE** – Address range is still in use
- **HSA_STATUS_ERROR** – Internal unexpected error

hsa_status_t **hsa_amd_vmem_handle_create**(*hsa_amd_memory_pool_t* pool, size_t size, *hsa_amd_memory_type_t* type, uint64_t flags, *hsa_amd_vmem_alloc_handle_t* *memory_handle)

Create a virtual memory handle.

Create a virtual memory handle within this pool `size` must be a aligned to allocation granule size for this memory pool, see HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_GRANULE To minimize internal memory fragmentation, align the size to the recommended allocation granule size, see HSA_AMD_MEMORY_POOL_INFO_RUNTIME_ALLOC_REC_GRANULE

Parameters

- **pool** – **[in]** memory to use
- **size** – **[in]** of the memory allocation
- **type** – **[in]** of memory
- **flags** – **[in]** - currently unsupported
- **memory_handle** – **[out]** - handle for the allocation

Return values

- **HSA_STATUS_SUCCESS** – memory allocated successfully
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – Invalid arguments
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – This memory pool does not support allocations
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Insufficient resources to allocate this memory

hsa_status_t **hsa_amd_vmem_handle_release**(*hsa_amd_vmem_alloc_handle_t* memory_handle)

Release a virtual memory handle.

Parameters

memory – **[in]** handle that was previously allocated

Return values

- **HSA_STATUS_SUCCESS** – Address range allocated successfully

- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – Invalid memory handle

hsa_status_t **hsa_amd_vmem_map**(void *va, size_t size, size_t in_offset, *hsa_amd_vmem_alloc_handle_t* memory_handle, uint64_t flags)

Map a virtual memory handle.

Map a virtual memory handle to a reserved address range. The virtual address requested must be within a previously reserved address range. `va` and (`va` + size) must be must be within (va + size) of the previous allocated address range. `size` must be equal to size of the `memory_handle` hsa_amd_vmem_set_access needs to be called to make the memory accessible to specific agents

Parameters

- **va** – **[in]** virtual address range where memory will be mapped
- **size** – **[in]** of memory mapping
- **in_offset** – **[in]** offset into memory. Currently unsupported
- **memory_handle** – **[in]** virtual memory handle to be mapped
- **flags.** – **[in]** Currently unsupported

Return values

- **HSA_STATUS_SUCCESS** – Memory mapped successfully
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – va, size or memory_handle are invalid
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Insufficient resources
- **HSA_STATUS_ERROR** – Unexpected internal error

hsa_status_t **hsa_amd_vmem_unmap**(void *va, size_t size)

Unmap a virtual memory handle.

Unmap previously mapped virtual address range

Parameters

- **va** – **[in]** virtual address range where memory will be mapped
- **size** – **[in]** of memory mapping

Return values

- **HSA_STATUS_SUCCESS** – Memory backing unmapped successfully
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – memory_handle is invalid
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – size is invalid
- **HSA_STATUS_ERROR** – Unexpected internal error

hsa_status_t **hsa_amd_vmem_set_access**(void *va, size_t size, const *hsa_amd_memory_access_desc_t* *desc, size_t desc_cnt)

Make a memory mapping accessible.

Make previously mapped virtual address accessible to specific agents. `size` must be equal to size of previously mapped virtual memory handle. Calling hsa_amd_vmem_set_access multiple times on the same `va`:

- Will overwrite permissions for agents specified in `desc`
- Will leave permissions unchanged for agents not specified in `desc`

Parameters

- **va** – **[in]** previously mapped virtual address
- **size** – **[in]** of memory mapping
- **desc** – **[in]** list of access permissions for each agent
- **desc_cnt** – **[in]** number of elements in desc

Return values

- **HSA_STATUS_SUCCESS** –
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – va, size or memory_handle are invalid
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – memory_handle is invalid
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Insufficient resources
- **HSA_STATUS_ERROR_INVALID_AGENT** – Invalid agent in desc
- **HSA_STATUS_ERROR** – Unexpected internal error

hsa_status_t **hsa_amd_vmem_get_access**(void *va, hsa_access_permission_t *perms, *hsa_agent_t* agent_handle)

Get current access permissions for memory mapping.

Get access permissions for memory mapping for specific agent.

Parameters

- **va** – **[in]** previously mapped virtual address
- **perms** – **[in]** current permissions
- **agent_handle** – **[in]** agent

Return values

- **HSA_STATUS_SUCCESS** –
- **HSA_STATUS_ERROR_INVALID_AGENT** – Invalid agent
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – va is not mapped or permissions never set for this agent
- **HSA_STATUS_ERROR** – Unexpected internal error

hsa_status_t **hsa_amd_vmem_export_shareable_handle**(int *dmabuf_fd, *hsa_amd_vmem_alloc_handle_t* handle, uint64_t flags)

Get an exportable shareable handle.

Get an exportable shareable handle for a memory_handle. This shareabl handle can then be used to re-create a virtual memory handle using hsa_amd_vmem_import_shareable_handle. The shareable handle can be transferred using mechanisms that support posix file descriptors Once all shareable handles are closed, the memory_handle is released.

Parameters

- **dmabuf_fd** – **[out]** shareable handle
- **handle** – **[in]** previously allocated virtual memory handle
- **flags** – **[in]** Currently unsupported

Return values

- **HSA_STATUS_SUCCESS** –
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – Invalid memory handle

- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Out of resources
- **HSA_STATUS_ERROR** – Unexpected internal error

hsa_status_t **hsa_amd_vmem_import_shareable_handle**(int dmabuf_fd, *hsa_amd_vmem_alloc_handle_t* *handle)

Import a shareable handle.

Import a shareable handle for a memory handle. Importing a shareable handle that has been closed and released results in undefined behavior.

Parameters

- **dmabuf_fd** – **[in]** shareable handle exported with hsa_amd_vmem_export_shareable_handle
- **handle** – **[out]** virtual memory handle

Return values

- **HSA_STATUS_SUCCESS** –
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – Invalid memory handle
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Out of resources
- **HSA_STATUS_ERROR** – Unexpected internal error

hsa_status_t **hsa_amd_vmem_retain_alloc_handle**(*hsa_amd_vmem_alloc_handle_t* *memory_handle, void *addr)

Returns memory handle for mapped memory.

Return a memory handle for previously mapped memory. The handle will be the same value of handle used to map the memory. The returned handle must be released with corresponding number of calls to hsa_amd_vmem_handle_release.

Parameters

- **memory_handle** – **[out]** memory handle for this mapped address
- **mapped** – **[in]** address

Return values

- **HSA_STATUS_SUCCESS** –
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – Invalid address

hsa_status_t **hsa_amd_vmem_get_alloc_properties_from_handle**(*hsa_amd_vmem_alloc_handle_t* memory_handle, *hsa_amd_memory_pool_t* *pool, *hsa_amd_memory_type_t* *type)

Returns the current allocation properties of a handle.

Returns the allocation properties of an existing handle

Parameters

- **memory_handle** – **[in]** memory handle to be queried
- **pool** – **[out]** memory pool that owns this handle
- **memory** – **[out]** type

Return values

- **HSA_STATUS_SUCCESS** –
- **HSA_STATUS_ERROR_INVALID_ALLOCATION** – Invalid memory_handle

struct **hsa_region_s**

#include <hsa.h> A memory region represents a block of virtual memory with certain properties. For example, the HSA runtime represents fine-grained memory in the global segment using a region. A region might be associated with more than one agent.

struct **hsa_amd_hdp_flush_s**

struct **hsa_amd_memory_pool_s**

#include <hsa_ext_amd.h> A memory pool encapsulates physical storage on an agent along with a memory access model.

A memory pool encapsulates a physical partition of an agent's memory system along with a memory access model. Division of a single memory system into separate pools allows querying each partition's access path properties (see *hsa_amd_agent_memory_pool_get_info*). Allocations from a pool are preferentially bound to that pool's physical partition. Binding to the pool's preferential physical partition may not be possible or persistent depending on the system's memory policy and/or state which is beyond the scope of HSA APIs.

For example, a multi-node NUMA memory system may be represented by multiple pool's with each pool providing size and access path information for the partition it represents. Allocations from a pool are preferentially bound to the pool's partition (which in this example is a NUMA node) while following its memory access model. The actual placement may vary or migrate due to the system's NUMA policy and state, which is beyond the scope of HSA APIs.

struct **hsa_pitched_ptr_s**

struct **hsa_amd_memory_pool_link_info_s**

#include <hsa_ext_amd.h> Link properties when accessing the memory pool from the specified agent.

struct **hsa_amd_pointer_info_s**

#include <hsa_ext_amd.h> Describes a memory allocation known to ROCr. Within a ROCr major version this structure can only grow.

struct **hsa_amd_ipc_memory_s**

#include <hsa_ext_amd.h> 256-bit process independent identifier for a ROCr shared memory allocation.

struct **hsa_amd_svm_attribute_pair_s**

struct **hsa_amd_vmem_alloc_handle_s**

#include <hsa_ext_amd.h> Struct containing an opaque handle to a memory allocation handle.

struct **hsa_amd_memory_access_desc_s**

4.9 Queues

enum **hsa_queue_type_t**

Queue type. Intended to be used for dynamic queue protocol determination.

Values:

enumerator **HSA_QUEUE_TYPE_MULTI**

Queue supports multiple producers. Use of multiproducer queue mechanics is required.

enumerator **HSA_QUEUE_TYPE_SINGLE**

Queue only supports a single producer. In some scenarios, the application may want to limit the submission of AQL packets to a single agent. Queues that support a single producer may be more efficient than queues supporting multiple producers. Use of multiproducer queue mechanics is not supported.

enumerator **HSA_QUEUE_TYPE_COOPERATIVE**

Queue supports multiple producers and cooperative dispatches. Cooperative dispatches are able to use GWS synchronization. Queues of this type may be limited in number. The runtime may return the same queue to serve multiple *hsa_queue_create* calls when this type is given. Callers must inspect the returned queue to discover queue size. Queues of this type are reference counted and require a matching number of *hsa_queue_destroy* calls to release. Use of multiproducer queue mechanics is required. See *HSA_AMD_AGENT_INFO_COOPERATIVE_QUEUES* to query agent support for this type.

enum **hsa_queue_feature_t**

Queue features.

Values:

enumerator **HSA_QUEUE_FEATURE_KERNEL_DISPATCH**

Queue supports kernel dispatch packets.

enumerator **HSA_QUEUE_FEATURE_AGENT_DISPATCH**

Queue supports agent dispatch packets.

enum **hsa_amd_queue_priority_s**

Per-queue dispatch and wavefront scheduling priority.

Values:

enumerator **HSA_AMD_QUEUE_PRIORITY_LOW**

enumerator **HSA_AMD_QUEUE_PRIORITY_NORMAL**

enumerator **HSA_AMD_QUEUE_PRIORITY_HIGH**

enum **hsa_queue_info_attribute_t**

Values:

enumerator **HSA_AMD_QUEUE_INFO_AGENT**

enumerator **HSA_AMD_QUEUE_INFO_DOORBELL_ID**

enum **hsa_amd_log_flag_s**

logging types

Values:

enumerator **HSA_AMD_LOG_FLAG_BLIT_KERNEL_PKTS**

typedef uint32_t **hsa_queue_type32_t**

A fixed-size type used to represent *hsa_queue_type_t* constants.

typedef struct *hsa_queue_s* **hsa_queue_t**

User mode queue.

The queue structure is read-only and allocated by the HSA runtime, but agents can directly modify the contents of the buffer pointed by *base_address*, or use HSA runtime APIs to access the doorbell signal.

typedef enum *hsa_amd_queue_priority_s* **hsa_amd_queue_priority_t**

Per-queue dispatch and wavefront scheduling priority.

typedef enum *hsa_amd_log_flag_s* **hsa_amd_log_flag_t**

logging types

hsa_status_t HSA_API hsa_queue_create (hsa_agent_t agent, uint32_t size, hsa_queue_type32_t type, void(*callback)(hsa_status_t status, hsa_queue_t *source, void *data), void *data, uint32_t private_segment_size, uint32_t group_segment_size, hsa_queue_t **queue)

Create a user mode queue.

The HSA runtime creates the queue structure, the underlying packet buffer, the completion signal, and the write and read indexes. The initial value of the write and read indexes is 0. The type of every packet in the buffer is initialized to *HSA_PACKET_TYPE_INVALID*.

The application should only rely on the error code returned to determine if the queue is valid.

Parameters

- **agent** – **[in]** Agent where to create the queue.
- **size** – **[in]** Number of packets the queue is expected to hold. Must be a power of 2 between 1 and the value of *HSA_AGENT_INFO_QUEUE_MAX_SIZE* in `agent`. The size of the newly created queue is the maximum of `size` and the value of *HSA_AGENT_INFO_QUEUE_MIN_SIZE* in `agent`.
- **type** – **[in]** Type of the queue, a bitwise OR of hsa_queue_type_t values. If the value of *HSA_AGENT_INFO_QUEUE_TYPE* in `agent` is *HSA_QUEUE_TYPE_SINGLE*, then `type` must also be *HSA_QUEUE_TYPE_SINGLE*.
- **callback** – **[in]** Callback invoked by the HSA runtime for every asynchronous event related to the newly created queue. May be NULL. The HSA runtime passes three arguments to the

callback: a code identifying the event that triggered the invocation, a pointer to the queue where the event originated, and the application data.

- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.
- **private_segment_size** – **[in]** Hint indicating the maximum expected private segment usage per work-item, in bytes. There may be performance degradation if the application places a kernel dispatch packet in the queue and the corresponding private segment usage exceeds `private_segment_size`. If the application does not want to specify any particular value for this argument, `private_segment_size` must be UINT32_MAX. If the queue does not support kernel dispatch packets, this argument is ignored.
- **group_segment_size** – **[in]** Hint indicating the maximum expected group segment usage per work-group, in bytes. There may be performance degradation if the application places a kernel dispatch packet in the queue and the corresponding group segment usage exceeds `group_segment_size`. If the application does not want to specify any particular value for this argument, `group_segment_size` must be UINT32_MAX. If the queue does not support kernel dispatch packets, this argument is ignored.
- **queue** – **[out]** Memory location where the HSA runtime stores a pointer to the newly created queue.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_QUEUE_CREATION** – `agent` does not support queues of the given type.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `size` is not a power of two, `size` is 0, `type` is an invalid queue type, or `queue` is NULL.

hsa_status_t HSA_API hsa_soft_queue_create (hsa_region_t region, uint32_t size, hsa_queue_type32_t type, uint32_t features, hsa_signal_t doorbell_signal, hsa_queue_t **queue)

Create a queue for which the application or a kernel is responsible for processing the AQL packets.

The application can use this function to create queues where AQL packets are not parsed by the packet processor associated with an agent, but rather by a unit of execution running on that agent (for example, a thread in the host application).

The application is responsible for ensuring that all the producers and consumers of the resulting queue can access the provided doorbell signal and memory region. The application is also responsible for ensuring that the unit of execution processing the queue packets supports the indicated features (AQL packet types).

When the queue is created, the HSA runtime allocates the packet buffer using `region`, and the write and read indexes. The initial value of the write and read indexes is 0, and the type of every packet in the buffer is initialized to *HSA_PACKET_TYPE_INVALID*. The value of the *size*, *type*, *features*, and *doorbell_signal* fields in the returned queue match the values passed by the application.

Parameters

- **region** – **[in]** Memory region that the HSA runtime should use to allocate the AQL packet buffer and any other queue metadata.

- **size** – **[in]** Number of packets the queue is expected to hold. Must be a power of 2 greater than 0.
- **type** – **[in]** Queue type.
- **features** – **[in]** Supported queue features. This is a bit-field of *hsa_queue_feature_t* values.
- **doorbell_signal** – **[in]** Doorbell signal that the HSA runtime must associate with the returned queue. The signal handle must not be 0.
- **queue** – **[out]** Memory location where the HSA runtime stores a pointer to the newly created queue. The application should not rely on the value returned for this argument but only in the status code to determine if the queue is valid. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `size` is not a power of two, `size` is 0, `type` is an invalid queue type, the doorbell signal handle is 0, or `queue` is NULL.

hsa_status_t HSA_API hsa_queue_destroy (hsa_queue_t *queue)

Destroy a user mode queue.

When a queue is destroyed, the state of the AQL packets that have not been yet fully processed (their completion phase has not finished) becomes undefined. It is the responsibility of the application to ensure that all pending queue operations are finished if their results are required.

The resources allocated by the HSA runtime during queue creation (queue structure, ring buffer, doorbell signal) are released. The queue should not be accessed after being destroyed.

Parameters

queue – **[in]** Pointer to a queue created using *hsa_queue_create*.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_QUEUE** – The queue is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `queue` is NULL.

hsa_status_t HSA_API hsa_queue_inactivate (hsa_queue_t *queue)

Inactivate a queue.

Inactivating the queue aborts any pending executions and prevent any new packets from being processed. Any more packets written to the queue once it is inactivated will be ignored by the packet processor.

Parameters

queue – **[in]** Pointer to a queue.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

- **HSA_STATUS_ERROR_INVALID_QUEUE** – The queue is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – queue is NULL.

uint64_t HSA_API HSA_DEPRECATED hsa_queue_load_read_index_acquire (const hsa_queue_t *queue)

Atomically load the read index of a queue.

Deprecated:

Renamed as *hsa_queue_load_read_index_scacquire*.

Parameters

queue – **[in]** Pointer to a queue.

Returns

Read index of the queue pointed by queue.

uint64_t HSA_API hsa_queue_load_read_index_scacquire (const hsa_queue_t *queue)

Atomically load the read index of a queue.

Parameters

queue – **[in]** Pointer to a queue.

Returns

Read index of the queue pointed by queue.

uint64_t HSA_API hsa_queue_load_read_index_relaxed (const hsa_queue_t *queue)

Atomically load the read index of a queue.

Parameters

queue – **[in]** Pointer to a queue.

Returns

Read index of the queue pointed by queue.

uint64_t HSA_API HSA_DEPRECATED hsa_queue_load_write_index_acquire (const hsa_queue_t *queue)

Atomically load the write index of a queue.

Deprecated:

Renamed as *hsa_queue_load_write_index_scacquire*.

Parameters

queue – **[in]** Pointer to a queue.

Returns

Write index of the queue pointed by queue.

uint64_t HSA_API hsa_queue_load_write_index_scacquire (const hsa_queue_t *queue)

Atomically load the write index of a queue.

Parameters

queue – **[in]** Pointer to a queue.

Returns

Write index of the queue pointed by queue.

uint64_t HSA_API hsa_queue_load_write_index_relaxed (const hsa_queue_t *queue)

Atomically load the write index of a queue.

Parameters

queue – **[in]** Pointer to a queue.

Returns

Write index of the queue pointed by queue.

void HSA_API hsa_queue_store_write_index_relaxed (const hsa_queue_t *queue, uint64_t value)

Atomically set the write index of a queue.

It is recommended that the application uses this function to update the write index when there is a single agent submitting work to the queue (the queue type is *HSA_QUEUE_TYPE_SINGLE*).

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to assign to the write index.

void HSA_API HSA_DEPRECATED hsa_queue_store_write_index_release (const hsa_queue_t *queue, uint64_t value)

Atomically set the write index of a queue.

Deprecated:

Renamed as *hsa_queue_store_write_index_screlease*.

It is recommended that the application uses this function to update the write index when there is a single agent submitting work to the queue (the queue type is *HSA_QUEUE_TYPE_SINGLE*).

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to assign to the write index.

void HSA_API hsa_queue_store_write_index_screlease (const hsa_queue_t *queue, uint64_t value)

Atomically set the write index of a queue.

It is recommended that the application uses this function to update the write index when there is a single agent submitting work to the queue (the queue type is *HSA_QUEUE_TYPE_SINGLE*).

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to assign to the write index.

uint64_t HSA_API HSA_DEPRECATED hsa_queue_cas_write_index_acq_rel (const hsa_queue_t *queue, uint64_t expected, uint64_t value)

Atomically set the write index of a queue if the observed value is equal to the expected value. The application can inspect the returned value to determine if the replacement was done.

Deprecated:

Renamed as *hsa_queue_cas_write_index_scacq_screl*.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **expected** – **[in]** Expected value.
- **value** – **[in]** Value to assign to the write index if `expected` matches the observed write index. Must be greater than `expected`.

Returns

Previous value of the write index.

uint64_t HSA_API hsa_queue_cas_write_index_scacq_screl (const hsa_queue_t *queue, uint64_t expected, uint64_t value)

Atomically set the write index of a queue if the observed value is equal to the expected value. The application can inspect the returned value to determine if the replacement was done.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **expected** – **[in]** Expected value.
- **value** – **[in]** Value to assign to the write index if `expected` matches the observed write index. Must be greater than `expected`.

Returns

Previous value of the write index.

uint64_t HSA_API HSA_DEPRECATED hsa_queue_cas_write_index_acquire (const hsa_queue_t *queue, uint64_t expected, uint64_t value)

Atomically set the write index of a queue if the observed value is equal to the expected value. The application can inspect the returned value to determine if the replacement was done.

Deprecated:

Renamed as *hsa_queue_cas_write_index_scacquire*.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **expected** – **[in]** Expected value.
- **value** – **[in]** Value to assign to the write index if `expected` matches the observed write index. Must be greater than `expected`.

Returns

Previous value of the write index.

uint64_t HSA_API hsa_queue_cas_write_index_scacquire (const hsa_queue_t *queue, uint64_t expected, uint64_t value)

Atomically set the write index of a queue if the observed value is equal to the expected value. The application can inspect the returned value to determine if the replacement was done.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **expected** – **[in]** Expected value.
- **value** – **[in]** Value to assign to the write index if `expected` matches the observed write index. Must be greater than `expected`.

Returns
Previous value of the write index.

uint64_t HSA_API hsa_queue_cas_write_index_relaxed (const hsa_queue_t *queue, uint64_t expected, uint64_t value)

Atomically set the write index of a queue if the observed value is equal to the expected value. The application can inspect the returned value to determine if the replacement was done.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **expected** – **[in]** Expected value.
- **value** – **[in]** Value to assign to the write index if `expected` matches the observed write index. Must be greater than `expected`.

Returns
Previous value of the write index.

uint64_t HSA_API HSA_DEPRECATED hsa_queue_cas_write_index_release (const hsa_queue_t *queue, uint64_t expected, uint64_t value)

Atomically set the write index of a queue if the observed value is equal to the expected value. The application can inspect the returned value to determine if the replacement was done.

Deprecated:
Renamed as *hsa_queue_cas_write_index_screlease*.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **expected** – **[in]** Expected value.
- **value** – **[in]** Value to assign to the write index if `expected` matches the observed write index. Must be greater than `expected`.

Returns
Previous value of the write index.

uint64_t HSA_API hsa_queue_cas_write_index_screlease (const hsa_queue_t *queue, uint64_t expected, uint64_t value)

Atomically set the write index of a queue if the observed value is equal to the expected value. The application can inspect the returned value to determine if the replacement was done.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **expected** – **[in]** Expected value.
- **value** – **[in]** Value to assign to the write index if `expected` matches the observed write index. Must be greater than `expected`.

Returns
Previous value of the write index.

`uint64_t HSA_API HSA_DEPRECATED hsa_queue_add_write_index_acq_rel (const hsa_queue_t *queue, uint64_t value)`

Atomically increment the write index of a queue by an offset.

Deprecated:

Renamed as *hsa_queue_add_write_index_scacq_screl*.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to add to the write index.

Returns
Previous value of the write index.

`uint64_t HSA_API hsa_queue_add_write_index_scacq_screl (const hsa_queue_t *queue, uint64_t value)`

Atomically increment the write index of a queue by an offset.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to add to the write index.

Returns
Previous value of the write index.

`uint64_t HSA_API HSA_DEPRECATED hsa_queue_add_write_index_acquire (const hsa_queue_t *queue, uint64_t value)`

Atomically increment the write index of a queue by an offset.

Deprecated:

Renamed as *hsa_queue_add_write_index_scacquire*.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to add to the write index.

Returns
Previous value of the write index.

`uint64_t HSA_API hsa_queue_add_write_index_scacquire (const hsa_queue_t *queue, uint64_t value)`

Atomically increment the write index of a queue by an offset.

Parameters

- **queue** – **[in]** Pointer to a queue.

- **value** – **[in]** Value to add to the write index.

Returns
Previous value of the write index.

uint64_t HSA_API hsa_queue_add_write_index_relaxed (const hsa_queue_t *queue, uint64_t value)

Atomically increment the write index of a queue by an offset.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to add to the write index.

Returns
Previous value of the write index.

uint64_t HSA_API HSA_DEPRECATED hsa_queue_add_write_index_release (const hsa_queue_t *queue, uint64_t value)

Atomically increment the write index of a queue by an offset.

Deprecated:
Renamed as *hsa_queue_add_write_index_screlease*.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to add to the write index.

Returns
Previous value of the write index.

uint64_t HSA_API hsa_queue_add_write_index_screlease (const hsa_queue_t *queue, uint64_t value)

Atomically increment the write index of a queue by an offset.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to add to the write index.

Returns
Previous value of the write index.

void HSA_API hsa_queue_store_read_index_relaxed (const hsa_queue_t *queue, uint64_t value)

Atomically set the read index of a queue.

Modifications of the read index are not allowed and result in undefined behavior if the queue is associated with an agent for which only the corresponding packet processor is permitted to update the read index.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to assign to the read index.

void HSA_API HSA_DEPRECATED hsa_queue_store_read_index_release (const hsa_queue_t *queue, uint64_t value)

Atomically set the read index of a queue.

Deprecated:

Renamed as *hsa_queue_store_read_index_screlease*.

Modifications of the read index are not allowed and result in undefined behavior if the queue is associated with an agent for which only the corresponding packet processor is permitted to update the read index.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to assign to the read index.

void HSA_API hsa_queue_store_read_index_screlease (const hsa_queue_t *queue, uint64_t value)

Atomically set the read index of a queue.

Modifications of the read index are not allowed and result in undefined behavior if the queue is associated with an agent for which only the corresponding packet processor is permitted to update the read index.

Parameters

- **queue** – **[in]** Pointer to a queue.
- **value** – **[in]** Value to assign to the read index.

hsa_status_t HSA_API hsa_amd_queue_cu_set_mask (const hsa_queue_t *queue, uint32_t num_cu_mask_count, const uint32_t *cu_mask)

Set a queue's CU affinity mask.

Enables the queue to run on only selected CUs. The given mask is combined by bitwise AND with any device wide mask in HSA_CU_MASK before being applied. If num_cu_mask_count is 0 then the request is interpreted as a request to enable all CUs and no cu_mask array need be given.

Parameters

- **queue** – **[in]** A pointer to HSA queue.
- **num_cu_mask_count** – **[in]** Size of CUMask bit array passed in, in bits.
- **cu_mask** – **[in]** Bit-vector representing the CU mask.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_CU_MASK_REDUCED** – The function was successfully executed but the given mask attempted to enable a CU which was disabled by HSA_CU_MASK. CUs disabled by HSA_CU_MASK remain disabled.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_QUEUE** – queue is NULL or invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – num_cu_mask_count is not a multiple of 32 or num_cu_mask_count is not 0 and cu_mask is NULL. Devices with work group processors must even-index contiguous pairwise CU enable e.g. 0x33(b'110011) is valid while 0x5(0x101) and 0x6(b'0110) are invalid.

hsa_status_t HSA_API hsa_amd_queue_cu_get_mask (const hsa_queue_t *queue, uint32_t num_cu_mask_count, uint32_t *cu_mask)

Retrieve a queue's CU affinity mask.

Returns the first num_cu_mask_count bits of a queue's CU mask. Ensure that num_cu_mask_count is at least as large as HSA_AMD_AGENT_INFO_COMPUTE_UNIT_COUNT to retrieve the entire mask.

Parameters

- **queue** – **[in]** A pointer to HSA queue.
- **num_cu_mask_count** – **[in]** Size of CUMask bit array passed in, in bits.
- **cu_mask** – **[out]** Bit-vector representing the CU mask.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_QUEUE** – queue is NULL or invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – num_cu_mask_count is 0, not a multiple of 32 or cu_mask is NULL.

hsa_status_t HSA_API hsa_amd_queue_set_priority (hsa_queue_t *queue, hsa_amd_queue_priority_t priority)

Modifies the dispatch and wavefront scheduling prioirty for a given compute queue. The default is HSA_AMD_QUEUE_PRIORITY_NORMAL.

Parameters

- **queue** – **[in]** Compute queue to apply new priority to.
- **priority** – **[in]** Priority to associate with queue.

Return values

- **HSA_STATUS_SUCCESS** – if priority was changed successfully.
- **HSA_STATUS_ERROR_INVALID_QUEUE** – if queue is not a valid compute queue handle.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – if priority is not a valid value from hsa_amd_queue_priority_t.

hsa_status_t HSA_API hsa_amd_agent_set_async_scratch_limit (hsa_agent_t agent, size_t threshold)

Set the asynchronous scratch limit threshold on all the queues for this agent. Dispatches that are enqueued on HW queues on this agent that are smaller than threshold will not result in a scratch use-once method.

Increasing this threshold will only increase the internal limit and not cause immediate allocation of additional scratch memory. Decreasing this threshold will result in a release in scratch memory on queues where the current amount of allocated scratch exceeds the new limit.

This API is only supported on devices that support asynchronous scratch reclaim.

Parameters

- **agent** – **[in]** A valid agent.
- **threshold** – **[in]** Threshold size in bytes

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – This agent does not support asynchronous scratch reclaim

hsa_status_t **hsa_amd_queue_get_info**(*hsa_queue_t* *queue, *hsa_queue_info_attribute_t* attribute, void *value)

hsa_status_t **hsa_amd_enable_logging**(uint8_t *flags, void *file)

Enable logging via external file If this function is called multiple times, the last call to this function will overwrite the previous `flags` and `file`.

Parameters

- **flags** – **[in]** is used to filter types of logging. Type is uint8_t[8]. Can be set using the hsa_flag_set64 macro. Setting `flags` to 0 will disable logging.
- **file** – **[in]** file stream to output logging. If file is NULL, prints are sent to stderr.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

struct **hsa_queue_s**

#include <*hsa.h*> User mode queue.

The queue structure is read-only and allocated by the HSA runtime, but agents can directly modify the contents of the buffer pointed by *base_address*, or use HSA runtime APIs to access the doorbell signal.

4.10 Runtime notifications

enum **hsa_amd_signal_attribute_t**

Signal attribute flags.

Values:

enumerator **HSA_AMD_SIGNAL_AMD_GPU_ONLY**

Signal will only be consumed by AMD GPUs. Limits signal consumption to AMD GPU agents only. Ignored if `num_consumers` is not zero (all agents).

enumerator **HSA_AMD_SIGNAL_IPC**

Signal may be used for interprocess communication. IPC signals can be read, written, and waited on from any process. Profiling using an IPC enabled signal is only supported in a single process at a time. Producing profiling data in one process and consuming it in another process is undefined.

enum **hsa_amd_event_type_s**

GPU system event type.

Values:

enumerator **HSA_AMD_GPU_MEMORY_FAULT_EVENT**

enumerator **HSA_AMD_GPU_HW_EXCEPTION_EVENT**

enumerator **HSA_AMD_GPU_MEMORY_ERROR_EVENT**

enum **hsa_amd_memory_fault_reason_t**

Flags denoting the cause of a memory fault.

Values:

enumerator **HSA_AMD_MEMORY_FAULT_PAGE_NOT_PRESENT**

enumerator **HSA_AMD_MEMORY_FAULT_READ_ONLY**

enumerator **HSA_AMD_MEMORY_FAULT_NX**

enumerator **HSA_AMD_MEMORY_FAULT_HOST_ONLY**

enumerator **HSA_AMD_MEMORY_FAULT_DRAMECC**

enumerator **HSA_AMD_MEMORY_FAULT_IMPRECISE**

enumerator **HSA_AMD_MEMORY_FAULT_SRAMECC**

enumerator **HSA_AMD_MEMORY_FAULT_HANG**

enum **hsa_amd_memory_error_reason_t**

Flags denoting the cause of a memory error.

Values:

enumerator **HSA_AMD_MEMORY_ERROR_MEMORY_IN_USE**

enum **hsa_amd_hw_exception_reset_type_t**

Flags denoting the type of a HW exception.

Values:

enumerator **HSA_AMD_HW_EXCEPTION_RESET_TYPE_OTHER**

enum **hsa_amd_hw_exception_reset_cause_t**

Flags denoting the cause of a HW exception.

Values:

enumerator **HSA_AMD_HW_EXCEPTION_CAUSE_GPU_HANG**

enumerator **HSA_AMD_HW_EXCEPTION_CAUSE_ECC**

typedef bool (***hsa_amd_signal_handler**)(*hsa_signal_value_t* value, void *arg)

Asyncronous signal handler function type.

Type definition of callback function to be used with hsa_amd_signal_async_handler. This callback is invoked if the associated signal and condition are met. The callback receives the value of the signal which satisfied the associated wait condition and a user provided value. If the callback returns true then the callback will be called again if the associated signal and condition are satisfied again. If the callback returns false then it will not be called again.

Param value
[in] Contains the value of the signal observed by hsa_amd_signal_async_handler which caused the signal handler to be invoked.

Param arg
[in] Contains the user provided value given when the signal handler was registered with hsa_amd_signal_async_handler

Retval true
resumes monitoring the signal with this handler (as if calling hsa_amd_signal_async_handler again with identical parameters)

Retval false
stops monitoring the signal with this handler (handler will not be called again for this signal)

typedef *hsa_amd_ipc_memory_t* **hsa_amd_ipc_signal_t**

256-bit process independent identifier for a ROCr IPC signal.

typedef enum *hsa_amd_event_type_s* **hsa_amd_event_type_t**

GPU system event type.

typedef struct *hsa_amd_gpu_memory_fault_info_s* **hsa_amd_gpu_memory_fault_info_t**

AMD GPU memory fault event data.

typedef struct *hsa_amd_gpu_memory_error_info_s* **hsa_amd_gpu_memory_error_info_t**

AMD GPU memory error event data.

typedef struct *hsa_amd_gpu_hw_exception_info_s* **hsa_amd_gpu_hw_exception_info_t**

AMD GPU HW Exception event data.

typedef struct *hsa_amd_event_s* **hsa_amd_event_t**

AMD GPU event data passed to event handler.

typedef *hsa_status_t* (***hsa_amd_system_event_callback_t**)(const *hsa_amd_event_t* *event, void *data)

hsa_status_t HSA_API hsa_amd_signal_create (hsa_signal_value_t initial_value, uint32_t num_consumers, const hsa_agent_t *consumers, uint64_t attributes, hsa_signal_t *signal)

Create a signal with specific attributes.

Parameters

- **initial_value** – **[in]** Initial value of the signal.
- **num_consumers** – **[in]** Size of `consumers`. A value of 0 indicates that any agent might wait on the signal.
- **consumers** – **[in]** List of agents that might consume (wait on) the signal. If `num_consumers` is 0, this argument is ignored; otherwise, the HSA runtime might use the list to optimize the handling of the signal object. If an agent not listed in `consumers` waits on the returned signal, the behavior is undefined. The memory associated with `consumers` can be reused or freed after the function returns.
- **attributes** – **[in]** Requested signal attributes. Multiple signal attributes may be requested by combining them with bitwise OR. Requesting no attributes (`attributes` == 0) results in the same signal as would have been obtained via hsa_signal_create.
- **signal** – **[out]** Pointer to a memory location where the HSA runtime will store the newly created signal handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `signal` is NULL, `num_consumers` is greater than 0 but `consumers` is NULL, or `consumers` contains duplicates.

hsa_status_t **hsa_amd_signal_value_pointer**(*hsa_signal_t* signal, volatile *hsa_signal_value_t* **value_ptr)

Returns a pointer to the value of a signal.

Use of this API does not modify the lifetime of ::signal and any hsa_signal_value_t retrieved by this API has lifetime equal to that of ::signal.

This API is intended for partial interoperability with non-HSA compatible devices and should not be used where HSA interfaces are available.

Use of the signal value must comply with use restritions of ::signal. Use may result in data races if the operations performed are not platform atomic. Use with HSA_AMD_SIGNAL_AMD_GPU_ONLY or HSA_AMD_SIGNAL_IPC attributed signals is required.

Parameters

- **Signal** – **[in]** handle to extract the signal value pointer from.
- **Location** – **[out]** where the extracted signal value pointer will be placed.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_SIGNAL** – signal is not a valid hsa_signal_t
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – value_ptr is NULL.

hsa_status_t HSA_API hsa_amd_signal_async_handler (hsa_signal_t signal, hsa_signal_condition_t cond, hsa_signal_value_t value, hsa_amd_signal_handler handler, void *arg)

Register asynchronous signal handler function.

Allows registering a callback function and user provided value with a signal and wait condition. The callback will be invoked if the associated signal and wait condition are satisfied. Callbacks will be invoked serially but in an arbitrary order so callbacks should be independent of each other. After being invoked a callback may continue to wait for its associated signal and condition and, possibly, be invoked again. Or the callback may stop waiting. If the callback returns true then it will continue waiting and may be called again. If false then the callback will not wait again and will not be called again for the associated signal and condition. It is possible to register the same callback multiple times with the same or different signals and/or conditions. Each registration of the callback will be treated entirely independently.

Parameters

- **signal** – **[in]** hsa signal to be asynchronously monitored
- **cond** – **[in]** condition value to monitor for
- **value** – **[in]** signal value used in condition expression
- **handler** – **[in]** asynchronous signal handler invoked when signal's condition is met
- **arg** – **[in]** user provided value which is provided to handler when handler is invoked

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_SIGNAL** – signal is not a valid hsa_signal_t
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – handler is invalid (NULL)
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime is out of resources or blocking signals are not supported by the HSA driver component.

uint32_t HSA_API hsa_amd_signal_wait_all (uint32_t signal_count, hsa_signal_t *signals, hsa_signal_condition_t *conds, hsa_signal_value_t *values, uint64_t timeout_hint, hsa_wait_state_t wait_hint, hsa_signal_value_t *satisfying_values)

Wait for all signal-condition pairs to be satisfied.

Allows waiting for all of several signal and condition pairs to be satisfied. The function returns 0 if all signals met their conditions and -1 on a timeout. The value of each signal's satisfying value is returned in satisfying_value unless satisfying_value is nullptr. This function provides only relaxed memory semantics.

uint32_t HSA_API hsa_amd_signal_wait_any (uint32_t signal_count, hsa_signal_t *signals, hsa_signal_condition_t *conds, hsa_signal_value_t *values, uint64_t timeout_hint, hsa_wait_state_t wait_hint, hsa_signal_value_t *satisfying_value)

Wait for any signal-condition pair to be satisfied.

Allows waiting for any of several signal and conditions pairs to be satisfied. The function returns the index into the list of signals of the first satisfying signal-condition pair. The value of the satisfying signal's value is returned in satisfying_value unless satisfying_value is NULL. This function provides only relaxed memory semantics.

hsa_status_t HSA_API hsa_amd_ipc_signal_create (hsa_signal_t signal, hsa_amd_ipc_signal_t *handle)

Obtains an interprocess sharing handle for a signal. The handle is valid while the signal it references remains valid in any process. In general applications should confirm that the signal has been attached (via hsa_amd_ipc_signal_attach) in the remote process prior to destroying that signal in the local process. Repeated calls for the same signal may, but are not required to, return unique handles.

Parameters

- **signal** – **[in]** Signal created with attribute HSA_AMD_SIGNAL_IPC.
- **handle** – **[out]** Process independent identifier referencing the shared signal.

Return values

- **HSA_STATUS_SUCCESS** – `handle` is ready to use for interprocess sharing.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `signal` is not a valid signal created with attribute HSA_AMD_SIGNAL_IPC or handle is NULL.

hsa_status_t HSA_API hsa_amd_ipc_signal_attach (const hsa_amd_ipc_signal_t *handle, hsa_signal_t *signal)

Imports an IPC capable signal into the local process. If an IPC signal handle is attached multiple times in a process each attach may return a different signal handle. Each returned signal handle is refcounted and requires a matching number of calls to hsa_signal_destroy to release the shared signal.

Parameters

- **handle** – **[in]** Pointer to the identifier for the shared signal.
- **signal** – **[out]** Recieves a process local signal handle to the shared signal.

Return values

- **HSA_STATUS_SUCCESS** – if the signal is successfully imported.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – if HSA is not initialized
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – if there is a failure in allocating necessary resources
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `handle` is not valid.

hsa_status_t HSA_API hsa_amd_register_system_event_handler (hsa_amd_system_event_callback_t callback, void *data)

Register AMD GPU event handler.

Parameters

- **callback** – **[in]** Callback to be invoked when an event is triggered. The HSA runtime passes two arguments to the callback: `event` is defined per event by the HSA runtime, and `data` is the user data.
- **data** – **[in]** User data that is passed to `callback`. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The handler has been registered successfully.
- **HSA_STATUS_ERROR** – An event handler has already been registered.

- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `event` is invalid.

struct **hsa_amd_gpu_memory_fault_info_s**

#include <*hsa_ext_amd.h*> AMD GPU memory fault event data.

struct **hsa_amd_gpu_memory_error_info_s**

#include <*hsa_ext_amd.h*> AMD GPU memory error event data.

struct **hsa_amd_gpu_hw_exception_info_s**

#include <*hsa_ext_amd.h*> AMD GPU HW Exception event data.

struct **hsa_amd_event_s**

#include <*hsa_ext_amd.h*> AMD GPU event data passed to event handler.

4.11 Signals

enum **hsa_signal_condition_t**

Wait condition operator.

Values:

enumerator **HSA_SIGNAL_CONDITION_EQ**

The two operands are equal.

enumerator **HSA_SIGNAL_CONDITION_NE**

The two operands are not equal.

enumerator **HSA_SIGNAL_CONDITION_LT**

The first operand is less than the second operand.

enumerator **HSA_SIGNAL_CONDITION_GTE**

The first operand is greater than or equal to the second operand.

enum **hsa_wait_state_t**

State of the application thread during a signal wait.

Values:

enumerator **HSA_WAIT_STATE_BLOCKED**

The application thread may be rescheduled while waiting on the signal.

enumerator **HSA_WAIT_STATE_ACTIVE**

The application thread stays active while waiting on a signal.

typedef struct *hsa_signal_s* **hsa_signal_t**

Signal handle.

typedef int32_t **hsa_signal_value_t**

Signal value. The value occupies 32 bits in small machine mode, and 64 bits in large machine mode.

typedef struct *hsa_signal_group_s* **hsa_signal_group_t**

Group of signals.

hsa_status_t HSA_API hsa_signal_create (hsa_signal_value_t initial_value, uint32_t num_consumers, const hsa_agent_t *consumers, hsa_signal_t *signal)

Create a signal.

Parameters

- **initial_value** – **[in]** Initial value of the signal.
- **num_consumers** – **[in]** Size of `consumers`. A value of 0 indicates that any agent might wait on the signal.
- **consumers** – **[in]** List of agents that might consume (wait on) the signal. If `num_consumers` is 0, this argument is ignored; otherwise, the HSA runtime might use the list to optimize the handling of the signal object. If an agent not listed in `consumers` waits on the returned signal, the behavior is undefined. The memory associated with `consumers` can be reused or freed after the function returns.
- **signal** – **[out]** Pointer to a memory location where the HSA runtime will store the newly created signal handle. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `signal` is NULL, `num_consumers` is greater than 0 but `consumers` is NULL, or `consumers` contains duplicates.

hsa_status_t HSA_API hsa_signal_destroy (hsa_signal_t signal)

Destroy a signal previous created by *hsa_signal_create*.

Parameters

signal – **[in]** Signal.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_SIGNAL** – `signal` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – The handle in `signal` is 0.

hsa_signal_value_t HSA_API hsa_signal_load_scacquire (hsa_signal_t signal)

Atomically read the current value of a signal.

Parameters

signal – **[in]** Signal.

Returns
Value of the signal.

`hsa_signal_value_t HSA_API hsa_signal_load_relaxed (hsa_signal_t signal)`

Atomically read the current value of a signal.

Parameters
`signal` – **[in]** Signal.

Returns
Value of the signal.

`hsa_signal_value_t HSA_API HSA_DEPRECATED hsa_signal_load_acquire (hsa_signal_t signal)`

Atomically read the current value of a signal.

Deprecated:

Renamed as *hsa_signal_load_scacquire*.

Parameters
`signal` – **[in]** Signal.

Returns
Value of the signal.

`void HSA_API hsa_signal_store_relaxed (hsa_signal_t signal, hsa_signal_value_t value)`

Atomically set the value of a signal.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- `signal` – **[in]** Signal.
- `value` – **[in]** New signal value.

`void HSA_API hsa_signal_store_screlease (hsa_signal_t signal, hsa_signal_value_t value)`

Atomically set the value of a signal.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- `signal` – **[in]** Signal.
- `value` – **[in]** New signal value.

`void HSA_API HSA_DEPRECATED hsa_signal_store_release (hsa_signal_t signal, hsa_signal_value_t value)`

Atomically set the value of a signal.

Deprecated:

Renamed as *hsa_signal_store_screlease*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal.
- **value** – **[in]** New signal value.

void HSA_API hsa_signal_silent_store_relaxed (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal without necessarily notifying the the agents waiting on it.

The agents waiting on `signal` may not wake up even when the new value satisfies their wait condition. If the application wants to update the signal and there is no need to notify any agent, invoking this function can be more efficient than calling the non-silent counterpart.

Parameters

- **signal** – **[in]** Signal.
- **value** – **[in]** New signal value.

void HSA_API hsa_signal_silent_store_screlease (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal without necessarily notifying the the agents waiting on it.

The agents waiting on `signal` may not wake up even when the new value satisfies their wait condition. If the application wants to update the signal and there is no need to notify any agent, invoking this function can be more efficient than calling the non-silent counterpart.

Parameters

- **signal** – **[in]** Signal.
- **value** – **[in]** New signal value.

hsa_signal_value_t HSA_API hsa_signal_exchange_scacq_screl (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal and return its previous value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** New value.

Returns

Value of the signal prior to the exchange.

hsa_signal_value_t HSA_API HSA_DEPRECATED hsa_signal_exchange_acq_rel (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal and return its previous value.

Deprecated:

Renamed as *hsa_signal_exchange_scacq_screl*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** New value.

Returns

Value of the signal prior to the exchange.

hsa_signal_value_t HSA_API hsa_signal_exchange_scacquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal and return its previous value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** New value.

Returns

Value of the signal prior to the exchange.

hsa_signal_value_t HSA_API HSA_DEPRECATED hsa_signal_exchange_acquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal and return its previous value.

Deprecated:

Renamed as *hsa_signal_exchange_scacquire*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** New value.

Returns

Value of the signal prior to the exchange.

hsa_signal_value_t HSA_API hsa_signal_exchange_relaxed (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal and return its previous value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.

- **value** – **[in]** New value.

Returns
Value of the signal prior to the exchange.

hsa_signal_value_t HSA_API hsa_signal_exchange_screlease (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal and return its previous value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** New value.

Returns
Value of the signal prior to the exchange.

hsa_signal_value_t HSA_API HSA_DEPRECATED hsa_signal_exchange_release (hsa_signal_t signal, hsa_signal_value_t value)

Atomically set the value of a signal and return its previous value.

Deprecated:

Renamed as *hsa_signal_exchange_screlease*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** New value.

Returns
Value of the signal prior to the exchange.

hsa_signal_value_t HSA_API hsa_signal_cas_scacq_screl (hsa_signal_t signal, hsa_signal_value_t expected, hsa_signal_value_t value)

Atomically set the value of a signal if the observed value is equal to the expected value. The observed value is returned regardless of whether the replacement was done.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **expected** – **[in]** Value to compare with.
- **value** – **[in]** New value.

Returns
Observed value of the signal.

hsa_signal_value_t HSA_API HSA_DEPRECATED hsa_signal_cas_acq_rel (hsa_signal_t signal, hsa_signal_value_t expected, hsa_signal_value_t value)

Atomically set the value of a signal if the observed value is equal to the expected value. The observed value is returned regardless of whether the replacement was done.

Deprecated:

Renamed as *hsa_signal_cas_scacq_screl*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **expected** – **[in]** Value to compare with.
- **value** – **[in]** New value.

Returns

Observed value of the signal.

hsa_signal_value_t HSA_API hsa_signal_cas_scacquire (hsa_signal_t signal, hsa_signal_value_t expected, hsa_signal_value_t value)

Atomically set the value of a signal if the observed value is equal to the expected value. The observed value is returned regardless of whether the replacement was done.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **expected** – **[in]** Value to compare with.
- **value** – **[in]** New value.

Returns

Observed value of the signal.

hsa_signal_value_t HSA_API HSA_DEPRECATED hsa_signal_cas_acquire (hsa_signal_t signal, hsa_signal_value_t expected, hsa_signal_value_t value)

Atomically set the value of a signal if the observed value is equal to the expected value. The observed value is returned regardless of whether the replacement was done.

Deprecated:

Renamed as *hsa_signal_cas_scacquire*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **expected** – **[in]** Value to compare with.

- **value** – **[in]** New value.

Returns
Observed value of the signal.

hsa_signal_value_t HSA_API hsa_signal_cas_relaxed (hsa_signal_t signal, hsa_signal_value_t expected, hsa_signal_value_t value)

Atomically set the value of a signal if the observed value is equal to the expected value. The observed value is returned regardless of whether the replacement was done.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **expected** – **[in]** Value to compare with.
- **value** – **[in]** New value.

Returns
Observed value of the signal.

hsa_signal_value_t HSA_API hsa_signal_cas_screlease (hsa_signal_t signal, hsa_signal_value_t expected, hsa_signal_value_t value)

Atomically set the value of a signal if the observed value is equal to the expected value. The observed value is returned regardless of whether the replacement was done.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **expected** – **[in]** Value to compare with.
- **value** – **[in]** New value.

Returns
Observed value of the signal.

hsa_signal_value_t HSA_API HSA_DEPRECATED hsa_signal_cas_release (hsa_signal_t signal, hsa_signal_value_t expected, hsa_signal_value_t value)

Atomically set the value of a signal if the observed value is equal to the expected value. The observed value is returned regardless of whether the replacement was done.

Deprecated:

Renamed as *hsa_signal_cas_screlease*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **expected** – **[in]** Value to compare with.

- **value** – **[in]** New value.

Returns

Observed value of the signal.

void HSA_API hsa_signal_add_scacq_screl (hsa_signal_t signal, hsa_signal_value_t value)

Atomically increment the value of a signal by a given amount.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to add to the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_add_acq_rel (hsa_signal_t signal, hsa_signal_value_t value)

Atomically increment the value of a signal by a given amount.

Deprecated:

Renamed as *hsa_signal_add_scacq_screl*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to add to the value of the signal.

void HSA_API hsa_signal_add_scacquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically increment the value of a signal by a given amount.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to add to the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_add_acquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically increment the value of a signal by a given amount.

Deprecated:

Renamed as *hsa_signal_add_scacquire*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.

- **value** – **[in]** Value to add to the value of the signal.

void HSA_API hsa_signal_add_relaxed (hsa_signal_t signal, hsa_signal_value_t value)

Atomically increment the value of a signal by a given amount.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to add to the value of the signal.

void HSA_API hsa_signal_add_screlease (hsa_signal_t signal, hsa_signal_value_t value)

Atomically increment the value of a signal by a given amount.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to add to the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_add_release (hsa_signal_t signal, hsa_signal_value_t value)

Atomically increment the value of a signal by a given amount.

Deprecated:

Renamed as *hsa_signal_add_screlease*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to add to the value of the signal.

void HSA_API hsa_signal_subtract_scacq_screl (hsa_signal_t signal, hsa_signal_value_t value)

Atomically decrement the value of a signal by a given amount.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to subtract from the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_subtract_acq_rel (hsa_signal_t signal, hsa_signal_value_t value)

Atomically decrement the value of a signal by a given amount.

Deprecated:

Renamed as *hsa_signal_subtract_scacq_screl*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to subtract from the value of the signal.

void HSA_API hsa_signal_subtract_scacquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically decrement the value of a signal by a given amount.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to subtract from the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_subtract_acquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically decrement the value of a signal by a given amount.

Deprecated:

Renamed as *hsa_signal_subtract_scacquire*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to subtract from the value of the signal.

void HSA_API hsa_signal_subtract_relaxed (hsa_signal_t signal, hsa_signal_value_t value)

Atomically decrement the value of a signal by a given amount.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to subtract from the value of the signal.

void HSA_API hsa_signal_subtract_screlease (hsa_signal_t signal, hsa_signal_value_t value)

Atomically decrement the value of a signal by a given amount.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to subtract from the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_subtract_release (hsa_signal_t signal, hsa_signal_value_t value)

Atomically decrement the value of a signal by a given amount.

Deprecated:

Renamed as *hsa_signal_subtract_screlease*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to subtract from the value of the signal.

void HSA_API hsa_signal_and_scacq_screl (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise AND operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to AND with the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_and_acq_rel (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise AND operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_and_scacq_screl*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to AND with the value of the signal.

`void HSA_API hsa_signal_and_scacquire (hsa_signal_t signal, hsa_signal_value_t value)`

Atomically perform a bitwise AND operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to AND with the value of the signal.

`void HSA_API HSA_DEPRECATED hsa_signal_and_acquire (hsa_signal_t signal, hsa_signal_value_t value)`

Atomically perform a bitwise AND operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_and_scacquire*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to AND with the value of the signal.

`void HSA_API hsa_signal_and_relaxed (hsa_signal_t signal, hsa_signal_value_t value)`

Atomically perform a bitwise AND operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to AND with the value of the signal.

`void HSA_API hsa_signal_and_screlease (hsa_signal_t signal, hsa_signal_value_t value)`

Atomically perform a bitwise AND operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to AND with the value of the signal.

`void HSA_API HSA_DEPRECATED hsa_signal_and_release (hsa_signal_t signal, hsa_signal_value_t value)`

Atomically perform a bitwise AND operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_and_screlease*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to AND with the value of the signal.

void HSA_API hsa_signal_or_scacq_screl (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise OR operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to OR with the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_or_acq_rel (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise OR operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_or_scacq_screl*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to OR with the value of the signal.

void HSA_API hsa_signal_or_scacquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise OR operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to OR with the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_or_acquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise OR operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_or_scacquire*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to OR with the value of the signal.

void HSA_API hsa_signal_or_relaxed (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise OR operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to OR with the value of the signal.

void HSA_API hsa_signal_or_screlease (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise OR operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to OR with the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_or_release (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise OR operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_or_screlease*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to OR with the value of the signal.

void HSA_API hsa_signal_xor_scacq_screl (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise XOR operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to XOR with the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_xor_acq_rel (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise XOR operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_xor_scacq_screl*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to XOR with the value of the signal.

void HSA_API hsa_signal_xor_scacquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise XOR operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to XOR with the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_xor_acquire (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise XOR operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_xor_scacquire*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to XOR with the value of the signal.

void HSA_API hsa_signal_xor_relaxed (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise XOR operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.

- **value** – **[in]** Value to XOR with the value of the signal.

void HSA_API hsa_signal_xor_screlease (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise XOR operation between the value of a signal and a given value.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to XOR with the value of the signal.

void HSA_API HSA_DEPRECATED hsa_signal_xor_release (hsa_signal_t signal, hsa_signal_value_t value)

Atomically perform a bitwise XOR operation between the value of a signal and a given value.

Deprecated:

Renamed as *hsa_signal_xor_screlease*.

If the value of the signal is changed, all the agents waiting on `signal` for which `value` satisfies their wait condition are awakened.

Parameters

- **signal** – **[in]** Signal. If `signal` is a queue doorbell signal, the behavior is undefined.
- **value** – **[in]** Value to XOR with the value of the signal.

hsa_signal_value_t HSA_API hsa_signal_wait_scacquire (hsa_signal_t signal, hsa_signal_condition_t condition, hsa_signal_value_t compare_value, uint64_t timeout_hint, hsa_wait_state_t wait_state_hint)

Wait until a signal value satisfies a specified condition, or a certain amount of time has elapsed.

A wait operation can spuriously resume at any time sooner than the timeout (for example, due to system or other external factors) even when the condition has not been met.

The function is guaranteed to return if the signal value satisfies the condition at some point in time during the wait, but the value returned to the application might not satisfy the condition. The application must ensure that signals are used in such way that wait wakeup conditions are not invalidated before dependent threads have woken up.

When the wait operation internally loads the value of the passed signal, it uses the memory order indicated in the function name.

Parameters

- **signal** – **[in]** Signal.
- **condition** – **[in]** Condition used to compare the signal value with `compare_value`.
- **compare_value** – **[in]** Value to compare with.
- **timeout_hint** – **[in]** Maximum duration of the wait. Specified in the same unit as the system timestamp. The operation might block for a shorter or longer time even if the condition is not met. A value of UINT64_MAX indicates no maximum.

- **wait_state_hint** – **[in]** Hint used by the application to indicate the preferred waiting state. The actual waiting state is ultimately decided by HSA runtime and may not match the provided hint. A value of *HSA_WAIT_STATE_ACTIVE* may improve the latency of response to a signal update by avoiding rescheduling overhead.

Returns

Observed value of the signal, which might not satisfy the specified condition.

hsa_signal_value_t HSA_API hsa_signal_wait_relaxed (hsa_signal_t signal, hsa_signal_condition_t condition, hsa_signal_value_t compare_value, uint64_t timeout_hint, hsa_wait_state_t wait_state_hint)

Wait until a signal value satisfies a specified condition, or a certain amount of time has elapsed.

A wait operation can spuriously resume at any time sooner than the timeout (for example, due to system or other external factors) even when the condition has not been met.

The function is guaranteed to return if the signal value satisfies the condition at some point in time during the wait, but the value returned to the application might not satisfy the condition. The application must ensure that signals are used in such way that wait wakeup conditions are not invalidated before dependent threads have woken up.

When the wait operation internally loads the value of the passed signal, it uses the memory order indicated in the function name.

Parameters

- **signal** – **[in]** Signal.
- **condition** – **[in]** Condition used to compare the signal value with `compare_value`.
- **compare_value** – **[in]** Value to compare with.
- **timeout_hint** – **[in]** Maximum duration of the wait. Specified in the same unit as the system timestamp. The operation might block for a shorter or longer time even if the condition is not met. A value of UINT64_MAX indicates no maximum.
- **wait_state_hint** – **[in]** Hint used by the application to indicate the preferred waiting state. The actual waiting state is ultimately decided by HSA runtime and may not match the provided hint. A value of *HSA_WAIT_STATE_ACTIVE* may improve the latency of response to a signal update by avoiding rescheduling overhead.

Returns

Observed value of the signal, which might not satisfy the specified condition.

hsa_signal_value_t HSA_API HSA_DEPRECATED hsa_signal_wait_acquire (hsa_signal_t signal, hsa_signal_condition_t condition, hsa_signal_value_t compare_value, uint64_t timeout_hint, hsa_wait_state_t wait_state_hint)

Wait until a signal value satisfies a specified condition, or a certain amount of time has elapsed.

Deprecated:

Renamed as *hsa_signal_wait_scacquire*.

A wait operation can spuriously resume at any time sooner than the timeout (for example, due to system or other external factors) even when the condition has not been met.

The function is guaranteed to return if the signal value satisfies the condition at some point in time during the wait, but the value returned to the application might not satisfy the condition. The application must ensure that

signals are used in such way that wait wakeup conditions are not invalidated before dependent threads have woken up.

When the wait operation internally loads the value of the passed signal, it uses the memory order indicated in the function name.

Parameters

- **signal** – **[in]** Signal.
- **condition** – **[in]** Condition used to compare the signal value with `compare_value`.
- **compare_value** – **[in]** Value to compare with.
- **timeout_hint** – **[in]** Maximum duration of the wait. Specified in the same unit as the system timestamp. The operation might block for a shorter or longer time even if the condition is not met. A value of UINT64_MAX indicates no maximum.
- **wait_state_hint** – **[in]** Hint used by the application to indicate the preferred waiting state. The actual waiting state is ultimately decided by HSA runtime and may not match the provided hint. A value of *HSA_WAIT_STATE_ACTIVE* may improve the latency of response to a signal update by avoiding rescheduling overhead.

Returns

Observed value of the signal, which might not satisfy the specified condition.

hsa_status_t HSA_API hsa_signal_group_create (uint32_t num_signals, const hsa_signal_t *signals, uint32_t num_consumers, const hsa_agent_t *consumers, hsa_signal_group_t *signal_group)

Create a signal group.

Parameters

- **num_signals** – **[in]** Number of elements in `signals`. Must not be 0.
- **signals** – **[in]** List of signals in the group. The list must not contain any repeated elements. Must not be NULL.
- **num_consumers** – **[in]** Number of elements in `consumers`. Must not be 0.
- **consumers** – **[in]** List of agents that might consume (wait on) the signal group. The list must not contain repeated elements, and must be a subset of the set of agents that are allowed to wait on all the signals in the group. If an agent not listed in `consumers` waits on the returned group, the behavior is undefined. The memory associated with `consumers` can be reused or freed after the function returns. Must not be NULL.
- **signal_group** – **[out]** Pointer to newly created signal group. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – The HSA runtime failed to allocate the required resources.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `num_signals` is 0, `signals` is NULL, `num_consumers` is 0, `consumers` is NULL, or `signal_group` is NULL.

hsa_status_t HSA_API hsa_signal_group_destroy (hsa_signal_group_t signal_group)

Destroy a signal group previous created by *hsa_signal_group_create*.

Parameters

signal_group – **[in]** Signal group.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_SIGNAL_GROUP** – `signal_group` is invalid.

hsa_status_t HSA_API hsa_signal_group_wait_any_scacquire (hsa_signal_group_t signal_group, const hsa_signal_condition_t *conditions, const hsa_signal_value_t *compare_values, hsa_wait_state_t wait_state_hint, hsa_signal_t *signal, hsa_signal_value_t *value)

Wait until the value of at least one of the signals in a signal group satisfies its associated condition.

The function is guaranteed to return if the value of at least one of the signals in the group satisfies its associated condition at some point in time during the wait, but the signal value returned to the application may no longer satisfy the condition. The application must ensure that signals in the group are used in such way that wait wakeup conditions are not invalidated before dependent threads have woken up.

When this operation internally loads the value of the passed signal, it uses the memory order indicated in the function name.

Parameters

- **signal_group** – **[in]** Signal group.
- **conditions** – **[in]** List of conditions. Each condition, and the value at the same index in `compare_values`, is used to compare the value of the signal at that index in `signal_group` (the signal passed by the application to *hsa_signal_group_create* at that particular index). The size of `conditions` must not be smaller than the number of signals in `signal_group`; any extra elements are ignored. Must not be NULL.
- **compare_values** – **[in]** List of comparison values. The size of `compare_values` must not be smaller than the number of signals in `signal_group`; any extra elements are ignored. Must not be NULL.
- **wait_state_hint** – **[in]** Hint used by the application to indicate the preferred waiting state. The actual waiting state is decided by the HSA runtime and may not match the provided hint. A value of *HSA_WAIT_STATE_ACTIVE* may improve the latency of response to a signal update by avoiding rescheduling overhead.
- **signal** – **[out]** Signal in the group that satisfied the associated condition. If several signals satisfied their condition, the function can return any of those signals. Must not be NULL.
- **value** – **[out]** Observed value for `signal`, which might no longer satisfy the specified condition. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_INVALID_SIGNAL_GROUP** – `signal_group` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `conditions` is NULL, `compare_values` is NULL, `signal` is NULL, or `value` is NULL.

hsa_status_t HSA_API hsa_signal_group_wait_any_relaxed (hsa_signal_group_t signal_group, const hsa_signal_condition_t *conditions, const hsa_signal_value_t *compare_values, hsa_wait_state_t wait_state_hint, hsa_signal_t *signal, hsa_signal_value_t *value)

Wait until the value of at least one of the signals in a signal group satisfies its associated condition.

The function is guaranteed to return if the value of at least one of the signals in the group satisfies its associated condition at some point in time during the wait, but the signal value returned to the application may no longer satisfy the condition. The application must ensure that signals in the group are used in such way that wait wakeup conditions are not invalidated before dependent threads have woken up.

When this operation internally loads the value of the passed signal, it uses the memory order indicated in the function name.

Parameters

- **signal_group** – **[in]** Signal group.
- **conditions** – **[in]** List of conditions. Each condition, and the value at the same index in `compare_values`, is used to compare the value of the signal at that index in `signal_group` (the signal passed by the application to *hsa_signal_group_create* at that particular index). The size of `conditions` must not be smaller than the number of signals in `signal_group`; any extra elements are ignored. Must not be NULL.
- **compare_values** – **[in]** List of comparison values. The size of `compare_values` must not be smaller than the number of signals in `signal_group`; any extra elements are ignored. Must not be NULL.
- **wait_state_hint** – **[in]** Hint used by the application to indicate the preferred waiting state. The actual waiting state is decided by the HSA runtime and may not match the provided hint. A value of *HSA_WAIT_STATE_ACTIVE* may improve the latency of response to a signal update by avoiding rescheduling overhead.
- **signal** – **[out]** Signal in the group that satisfied the associated condition. If several signals satisfied their condition, the function can return any of those signals. Must not be NULL.
- **value** – **[out]** Observed value for `signal`, which might no longer satisfy the specified condition. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_INVALID_SIGNAL_GROUP** – `signal_group` is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `conditions` is NULL, `compare_values` is NULL, `signal` is NULL, or `value` is NULL.

struct **hsa_signal_s**

#include <hsa.h> Signal handle.

struct **hsa_signal_group_s**

#include <hsa.h> Group of signals.

4.12 System and agent information

enum **hsa_endianness_t**

Endianness. A convention used to interpret the bytes making up a data word.

Values:

enumerator **HSA_ENDIANNESS_LITTLE**

The least significant byte is stored in the smallest address.

enumerator **HSA_ENDIANNESS_BIG**

The most significant byte is stored in the smallest address.

enum **hsa_machine_model_t**

Machine model. A machine model determines the size of certain data types in HSA runtime and an agent.

Values:

enumerator **HSA_MACHINE_MODEL_SMALL**

Small machine model. Addresses use 32 bits.

enumerator **HSA_MACHINE_MODEL_LARGE**

Large machine model. Addresses use 64 bits.

enum **hsa_profile_t**

Profile. A profile indicates a particular level of feature support. For example, in the base profile the application must use the HSA runtime allocator to reserve shared virtual memory, while in the full profile any host pointer can be shared across all the agents.

Values:

enumerator **HSA_PROFILE_BASE**

Base profile.

enumerator **HSA_PROFILE_FULL**

Full profile.

enum **hsa_system_info_t**

System attributes.

Values:

enumerator **HSA_SYSTEM_INFO_VERSION_MAJOR**

Major version of the HSA runtime specification supported by the implementation. The type of this attribute is uint16_t.

enumerator **HSA_SYSTEM_INFO_VERSION_MINOR**

Minor version of the HSA runtime specification supported by the implementation. The type of this attribute is uint16_t.

enumerator **HSA_SYSTEM_INFO_TIMESTAMP**

Current timestamp. The value of this attribute monotonically increases at a constant rate. The type of this attribute is uint64_t.

enumerator **HSA_SYSTEM_INFO_TIMESTAMP_FREQUENCY**

Timestamp value increase rate, in Hz. The timestamp (clock) frequency is in the range 1-400MHz. The type of this attribute is uint64_t.

enumerator **HSA_SYSTEM_INFO_SIGNAL_MAX_WAIT**

Maximum duration of a signal wait operation. Expressed as a count based on the timestamp frequency. The type of this attribute is uint64_t.

enumerator **HSA_SYSTEM_INFO_ENDIANNESS**

Endianness of the system. The type of this attribute is *hsa_endianness_t*.

enumerator **HSA_SYSTEM_INFO_MACHINE_MODEL**

Machine model supported by the HSA runtime. The type of this attribute is *hsa_machine_model_t*.

enumerator **HSA_SYSTEM_INFO_EXTENSIONS**

Bit-mask indicating which extensions are supported by the implementation. An extension with an ID of `i` is supported if the bit at position `i` is set. The type of this attribute is uint8_t[128].

enumerator **HSA_AMD_SYSTEM_INFO_BUILD_VERSION**

String containing the ROCr build identifier.

enumerator **HSA_AMD_SYSTEM_INFO_SVM_SUPPORTED**

Returns true if hsa_amd_svm_* APIs are supported by the driver. The type of this attribute is bool.

enumerator **HSA_AMD_SYSTEM_INFO_SVM_ACCESSIBLE_BY_DEFAULT**

Returns true if all Agents have access to system allocated memory (such as that allocated by mmap, malloc, or new) by default. If false then system allocated memory may only be made SVM accessible to an Agent by declaration of accessibility with hsa_amd_svm_set_attributes. The type of this attribute is bool.

enumerator **HSA_AMD_SYSTEM_INFO_MWAITX_ENABLED**

Returns true if mwaitx is enabled on this system The type of this attribute is bool.

enumerator **HSA_AMD_SYSTEM_INFO_DMABUF_SUPPORTED**

Returns true if DMABUF APIs are supported by the driver. The type of this attribute is bool.

enumerator **HSA_AMD_SYSTEM_INFO_VIRTUAL_MEM_API_SUPPORTED**

Returns true if Virtual Memory APIs are supported by the driver. The type of this attribute is bool.

enumerator **HSA_AMD_SYSTEM_INFO_XNACK_ENABLED**

Returns true if XNACK is enabled on this system. The type of this attribute is bool.

enumerator **HSA_AMD_SYSTEM_INFO_EXT_VERSION_MAJOR**

Major version of the HSA runtime extension specification supported by the implementation. The type of this attribute is uint16_t.

enumerator **HSA_AMD_SYSTEM_INFO_EXT_VERSION_MINOR**

Minor version of the HSA runtime extension specification supported by the implementation. The type of this attribute is uint16_t.

enum **hsa_extension_t**

HSA extensions.

Values:

enumerator **HSA_EXTENSION_FINALIZER**

Finalizer extension.

enumerator **HSA_EXTENSION_IMAGES**

Images extension.

enumerator **HSA_EXTENSION_PERFORMANCE_COUNTERS**

Performance counter extension.

enumerator **HSA_EXTENSION_PROFILING_EVENTS**

Profiling events extension.

enumerator **HSA_EXTENSION_STD_LAST**

Extension count.

enumerator **HSA_AMD_FIRST_EXTENSION**

First AMD extension number.

enumerator **HSA_EXTENSION_AMD_PROFILER**

Profiler extension.

enumerator **HSA_EXTENSION_AMD_LOADER**

Loader extension.

enumerator **HSA_EXTENSION_AMD_AQLPROFILE**

AqlProfile extension.

enumerator **HSA_EXTENSION_AMD_PC_SAMPLING**

PC Sampling extension.

enumerator **HSA_AMD_LAST_EXTENSION**

Last AMD extension.

enum **hsa_agent_feature_t**

Agent features.

Values:

enumerator **HSA_AGENT_FEATURE_KERNEL_DISPATCH**

The agent supports AQL packets of kernel dispatch type. If this feature is enabled, the agent is also a kernel agent.

enumerator **HSA_AGENT_FEATURE_AGENT_DISPATCH**

The agent supports AQL packets of agent dispatch type.

enum **hsa_device_type_t**

Hardware device type.

Values:

enumerator **HSA_DEVICE_TYPE_CPU**

CPU device.

enumerator **HSA_DEVICE_TYPE_GPU**

GPU device.

enumerator **HSA_DEVICE_TYPE_DSP**

DSP device.

enumerator **HSA_DEVICE_TYPE_AIE**

AI Engine (AIE) device.

enum **hsa_default_float_rounding_mode_t**

Default floating-point rounding mode.

Values:

enumerator **HSA_DEFAULT_FLOAT_ROUNDING_MODE_DEFAULT**

Use a default floating-point rounding mode specified elsewhere.

enumerator **HSA_DEFAULT_FLOAT_ROUNDING_MODE_ZERO**

Operations that specify the default floating-point mode are rounded to zero by default.

enumerator **HSA_DEFAULT_FLOAT_ROUNDING_MODE_NEAR**

Operations that specify the default floating-point mode are rounded to the nearest representable number and that ties should be broken by selecting the value with an even least significant bit.

enum **hsa_agent_info_t**

Agent attributes.

Values:

enumerator **HSA_AGENT_INFO_NAME**

Agent name. The type of this attribute is a NUL-terminated char[64]. The name must be at most 63 characters long (not including the NUL terminator) and all array elements not used for the name must be NUL.

enumerator **HSA_AGENT_INFO_VENDOR_NAME**

Name of vendor. The type of this attribute is a NUL-terminated char[64]. The name must be at most 63 characters long (not including the NUL terminator) and all array elements not used for the name must be NUL.

enumerator **HSA_AGENT_INFO_FEATURE**

Agent capability. The type of this attribute is *hsa_agent_feature_t*.

enumerator **HSA_AGENT_INFO_MACHINE_MODEL**

Deprecated:

Query *HSA_ISA_INFO_MACHINE_MODELS* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Machine model supported by the agent. The type of this attribute is *hsa_machine_model_t*.

enumerator **HSA_AGENT_INFO_PROFILE**

Deprecated:

Query *HSA_ISA_INFO_PROFILES* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Profile supported by the agent. The type of this attribute is *hsa_profile_t*.

enumerator **HSA_AGENT_INFO_DEFAULT_FLOAT_ROUNDING_MODE**

Deprecated:

Query *HSA_ISA_INFO_DEFAULT_FLOAT_ROUNDING_MODES* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Default floating-point rounding mode. The type of this attribute is *hsa_default_float_rounding_mode_t*, but the value *HSA_DEFAULT_FLOAT_ROUNDING_MODE_DEFAULT* is not allowed.

enumerator **HSA_AGENT_INFO_BASE_PROFILE_DEFAULT_FLOAT_ROUNDING_MODES**

Deprecated:

Query *HSA_ISA_INFO_BASE_PROFILE_DEFAULT_FLOAT_ROUNDING_MODES* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

A bit-mask of *hsa_default_float_rounding_mode_t* values, representing the default floating-point rounding modes supported by the agent in the Base profile. The type of this attribute is uint32_t. The default floating-point rounding mode (*HSA_AGENT_INFO_DEFAULT_FLOAT_ROUNDING_MODE*) bit must not be set.

enumerator **HSA_AGENT_INFO_FAST_F16_OPERATION**

Deprecated:

Query *HSA_ISA_INFO_FAST_F16_OPERATION* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Flag indicating that the f16 HSAIL operation is at least as fast as the f32 operation in the current agent. The value of this attribute is undefined if the agent is not a kernel agent. The type of this attribute is bool.

enumerator **HSA_AGENT_INFO_WAVEFRONT_SIZE**

Deprecated:

Query *HSA_WAVEFRONT_INFO_SIZE* for a given wavefront and intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas* and the first wavefront enumerated by *hsa_isa_iterate_wavefronts* for that ISA.

Number of work-items in a wavefront. Must be a power of 2 in the range [1,256]. The value of this attribute is undefined if the agent is not a kernel agent. The type of this attribute is uint32_t.

enumerator **HSA_AGENT_INFO_WORKGROUP_MAX_DIM**

Deprecated:

Query *HSA_ISA_INFO_WORKGROUP_MAX_DIM* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Maximum number of work-items of each dimension of a work-group. Each maximum must be greater than 0. No maximum can exceed the value of *HSA_AGENT_INFO_WORKGROUP_MAX_SIZE*. The value of this attribute is undefined if the agent is not a kernel agent. The type of this attribute is uint16_t[3].

enumerator **HSA_AGENT_INFO_WORKGROUP_MAX_SIZE**

Deprecated:

Query *HSA_ISA_INFO_WORKGROUP_MAX_SIZE* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Maximum total number of work-items in a work-group. The value of this attribute is undefined if the agent is not a kernel agent. The type of this attribute is uint32_t.

enumerator **HSA_AGENT_INFO_GRID_MAX_DIM**

Deprecated:

Query *HSA_ISA_INFO_GRID_MAX_DIM* for a given intruction set architecture supported by the agent instead.

Maximum number of work-items of each dimension of a grid. Each maximum must be greater than 0, and must not be smaller than the corresponding value in *HSA_AGENT_INFO_WORKGROUP_MAX_DIM*. No maximum can exceed the value of *HSA_AGENT_INFO_GRID_MAX_SIZE*. The value of this attribute is undefined if the agent is not a kernel agent. The type of this attribute is hsa_dim3_t.

enumerator **HSA_AGENT_INFO_GRID_MAX_SIZE**

Deprecated:

Query *HSA_ISA_INFO_GRID_MAX_SIZE* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Maximum total number of work-items in a grid. The value of this attribute is undefined if the agent is not a kernel agent. The type of this attribute is uint32_t.

enumerator **HSA_AGENT_INFO_FBARRIER_MAX_SIZE**

Deprecated:

Query *HSA_ISA_INFO_FBARRIER_MAX_SIZE* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Maximum number of fbarriers per work-group. Must be at least 32. The value of this attribute is undefined if the agent is not a kernel agent. The type of this attribute is uint32_t.

enumerator **HSA_AGENT_INFO_QUEUES_MAX**

Deprecated:

The maximum number of queues is not statically determined.

Maximum number of queues that can be active (created but not destroyed) at one time in the agent. The type of this attribute is uint32_t.

enumerator **HSA_AGENT_INFO_QUEUE_MIN_SIZE**

Minimum number of packets that a queue created in the agent can hold. Must be a power of 2 greater than 0. Must not exceed the value of *HSA_AGENT_INFO_QUEUE_MAX_SIZE*. The type of this attribute is uint32_t.

enumerator **HSA_AGENT_INFO_QUEUE_MAX_SIZE**

Maximum number of packets that a queue created in the agent can hold. Must be a power of 2 greater than 0. The type of this attribute is uint32_t.

enumerator **HSA_AGENT_INFO_QUEUE_TYPE**

Type of a queue created in the agent. The type of this attribute is *hsa_queue_type32_t*.

enumerator **HSA_AGENT_INFO_NODE**

Deprecated:

NUMA information is not exposed anywhere else in the API.

Identifier of the NUMA node associated with the agent. The type of this attribute is uint32_t.

enumerator **HSA_AGENT_INFO_DEVICE**

Type of hardware device associated with the agent. The type of this attribute is *hsa_device_type_t*.

enumerator **HSA_AGENT_INFO_CACHE_SIZE**

Deprecated:

Query *hsa_agent_iterate_caches* to retrieve information about the caches present in a given agent.

Array of data cache sizes (L1..L4). Each size is expressed in bytes. A size of 0 for a particular level indicates that there is no cache information for that level. The type of this attribute is uint32_t[4].

enumerator **HSA_AGENT_INFO_ISA**

Deprecated:

An agent may support multiple instruction set architectures. See *hsa_agent_iterate_isas*. If more than one ISA is supported by the agent, the returned value corresponds to the first ISA enumerated by *hsa_agent_iterate_isas*.

Instruction set architecture of the agent. The type of this attribute is *hsa_isa_t*.

enumerator **HSA_AGENT_INFO_EXTENSIONS**

Bit-mask indicating which extensions are supported by the agent. An extension with an ID of `i` is supported if the bit at position `i` is set. The type of this attribute is uint8_t[128].

enumerator **HSA_AGENT_INFO_VERSION_MAJOR**

Major version of the HSA runtime specification supported by the agent. The type of this attribute is uint16_t.

enumerator **HSA_AGENT_INFO_VERSION_MINOR**

Minor version of the HSA runtime specification supported by the agent. The type of this attribute is uint16_t.

enumerator **HSA_AGENT_INFO_LAST**

This enum does not have a fixed underlying type, thus in C++ post D2338: If the enumeration type does not have a fixed underlying type, the value is unchanged if the original value is within the range of the enumeration values (9.7.1 [dcl.enum]), and otherwise, the behavior is undefined. Thus increase the range of this enum to encompass vendor extensions.

enum **hsa_exception_policy_t**

Exception policies applied in the presence of hardware exceptions.

Values:

enumerator **HSA_EXCEPTION_POLICY_BREAK**

If a hardware exception is detected, a work-item signals an exception.

enumerator **HSA_EXCEPTION_POLICY_DETECT**

If a hardware exception is detected, a hardware status bit is set.

enum **hsa_cache_info_t**

Cache attributes.

Values:

enumerator **HSA_CACHE_INFO_NAME_LENGTH**

The length of the cache name in bytes, not including the NUL terminator. The type of this attribute is uint32_t.

enumerator **HSA_CACHE_INFO_NAME**

Human-readable description. The type of this attribute is a NUL-terminated character array with the length equal to the value of *HSA_CACHE_INFO_NAME_LENGTH* attribute.

enumerator **HSA_CACHE_INFO_LEVEL**

Cache level. A L1 cache must return a value of 1, a L2 must return a value of 2, and so on. The type of this attribute is uint8_t.

enumerator **HSA_CACHE_INFO_SIZE**

Cache size, in bytes. A value of 0 indicates that there is no size information available. The type of this attribute is uint32_t.

typedef struct *hsa_agent_s* **hsa_agent_t**

Struct containing an opaque handle to an agent, a device that participates in the HSA memory model. An agent can submit AQL packets for execution, and may also accept AQL packets for execution (agent dispatch packets or kernel dispatch packets launching HSAIL-derived binaries).

typedef struct *hsa_cache_s* **hsa_cache_t**

Cache handle.

hsa_status_t HSA_API hsa_system_get_info (hsa_system_info_t attribute, void *value)

Get the current value of a system attribute.

Parameters

- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid system attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_extension_get_name (uint16_t extension, const char **name)

Query the name of a given extension.

Parameters

- **extension** – **[in]** Extension identifier. If the extension is not supported by the implementation (see *HSA_SYSTEM_INFO_EXTENSIONS*), the behavior is undefined.
- **name** – **[out]** Pointer to a memory location where the HSA runtime stores the extension name. The extension name is a NUL-terminated string.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `extension` is not a valid extension, or `name` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_system_extension_supported (uint16_t extension, uint16_t version_major, uint16_t version_minor, bool *result)

Query if a given version of an extension is supported by the HSA implementation.

Deprecated:

Parameters

- **extension** – **[in]** Extension identifier.
- **version_major** – **[in]** Major version number.
- **version_minor** – **[in]** Minor version number.
- **result** – **[out]** Pointer to a memory location where the HSA runtime stores the result of the check. The result is true if the specified version of the extension is supported, and false otherwise.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `extension` is not a valid extension, or `result` is NULL.

hsa_status_t HSA_API hsa_system_major_extension_supported (uint16_t extension, uint16_t version_major, uint16_t *version_minor, bool *result)

Query if a given version of an extension is supported by the HSA implementation. All minor versions from 0 up to the returned `version_minor` must be supported by the implementation.

Parameters

- **extension** – **[in]** Extension identifier.
- **version_major** – **[in]** Major version number.
- **version_minor** – **[out]** Minor version number.
- **result** – **[out]** Pointer to a memory location where the HSA runtime stores the result of the check. The result is true if the specified version of the extension is supported, and false otherwise.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `extension` is not a valid extension, or `version_minor` is NULL, or `result` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_system_get_extension_table (uint16_t extension, uint16_t version_major, uint16_t version_minor, void *table)

Retrieve the function pointers corresponding to a given version of an extension. Portable applications are expected to invoke the extension API using the returned function pointers.

Deprecated:

The application is responsible for verifying that the given version of the extension is supported by the HSA implementation (see *hsa_system_extension_supported*). If the given combination of extension, major version, and minor version is not supported by the implementation, the behavior is undefined.

Parameters

- **extension** – **[in]** Extension identifier.
- **version_major** – **[in]** Major version number for which to retrieve the function pointer table.
- **version_minor** – **[in]** Minor version number for which to retrieve the function pointer table.
- **table** – **[out]** Pointer to an application-allocated function pointer table that is populated by the HSA runtime. Must not be NULL. The memory associated with table can be reused or freed after the function returns.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `extension` is not a valid extension, or `table` is NULL.

hsa_status_t HSA_API hsa_system_get_major_extension_table (uint16_t extension, uint16_t version_major, size_t table_length, void *table)

Retrieve the function pointers corresponding to a given major version of an extension. Portable applications are expected to invoke the extension API using the returned function pointers.

The application is responsible for verifying that the given major version of the extension is supported by the HSA implementation (see *hsa_system_major_extension_supported*). If the given combination of extension and major version is not supported by the implementation, the behavior is undefined. Additionally if the length doesn't allow space for a full minor version, it is implementation defined if only some of the function pointers for that minor version get written.

Parameters

- **extension** – **[in]** Extension identifier.
- **version_major** – **[in]** Major version number for which to retrieve the function pointer table.
- **table_length** – **[in]** Size in bytes of the function pointer table to be populated. The implementation will not write more than this many bytes to the table.
- **table** – **[out]** Pointer to an application-allocated function pointer table that is populated by the HSA runtime. Must not be NULL. The memory associated with table can be reused or freed after the function returns.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `extension` is not a valid extension, or `table` is NULL.

hsa_status_t HSA_API hsa_agent_get_info (hsa_agent_t agent, hsa_agent_info_t attribute, void *value)

Get the current value of an attribute for a given agent.

Parameters

- **agent** – **[in]** A valid agent.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid agent attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_iterate_agents (hsa_status_t(*callback)(hsa_agent_t agent, void *data), void *data)

Iterate over the available agents, and invoke an application-defined callback on every iteration.

Parameters

- **callback** – **[in]** Callback to be invoked once per agent. The HSA runtime passes two arguments to the callback: the agent and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and *hsa_iterate_agents* returns that status value.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_agent_get_exception_policies (hsa_agent_t agent, hsa_profile_t profile, uint16_t *mask)

Retrieve the exception policy support for a given combination of agent and profile.

Deprecated:

Use *hsa_isa_get_exception_policies* for a given intruction set architecture supported by the agent instead. If more than one ISA is supported by the agent, this function uses the first value returned by *hsa_agent_iterate_isas*.

Parameters

- **agent** – **[in]** Agent.
- **profile** – **[in]** Profile.
- **mask** – **[out]** Pointer to a memory location where the HSA runtime stores a mask of *hsa_exception_policy_t* values. Must not be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `profile` is not a valid profile, or `mask` is NULL.

hsa_status_t HSA_API hsa_cache_get_info (hsa_cache_t cache, hsa_cache_info_t attribute, void *value)

Get the current value of an attribute for a given cache object.

Parameters

- **cache** – **[in]** Cache.
- **attribute** – **[in]** Attribute to query.
- **value** – **[out]** Pointer to an application-allocated buffer where to store the value of the attribute. If the buffer passed by the application is not large enough to hold the value of `attribute`, the behavior is undefined.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_CACHE** – The cache is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `attribute` is an invalid instruction set architecture attribute, or `value` is NULL.

hsa_status_t HSA_API hsa_agent_iterate_caches (hsa_agent_t agent, hsa_status_t(*callback)(hsa_cache_t cache, void *data), void *data)

Iterate over the memory caches of a given agent, and invoke an application-defined callback on every iteration.

Caches are visited in ascending order according to the value of the *HSA_CACHE_INFO_LEVEL* attribute.

Parameters

- **agent** – **[in]** A valid agent.
- **callback** – **[in]** Callback to be invoked once per cache that is present in the agent. The HSA runtime passes two arguments to the callback: the cache and the application data. If `callback` returns a status other than *HSA_STATUS_SUCCESS* for a particular iteration, the traversal stops and that value is returned.
- **data** – **[in]** Application data that is passed to `callback` on every iteration. May be NULL.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `callback` is NULL.

hsa_status_t HSA_API HSA_DEPRECATED hsa_agent_extension_supported (uint16_t extension, hsa_agent_t agent, uint16_t version_major, uint16_t version_minor, bool *result)

Query if a given version of an extension is supported by an agent.

Deprecated:

Parameters

- **extension** – **[in]** Extension identifier.
- **agent** – **[in]** Agent.
- **version_major** – **[in]** Major version number.
- **version_minor** – **[in]** Minor version number.
- **result** – **[out]** Pointer to a memory location where the HSA runtime stores the result of the check. The result is true if the specified version of the extension is supported, and false otherwise. The result must be false if *hsa_system_extension_supported* returns false for the same extension version.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `extension` is not a valid extension, or `result` is NULL.

hsa_status_t HSA_API hsa_agent_major_extension_supported (uint16_t extension, hsa_agent_t agent, uint16_t version_major, uint16_t *version_minor, bool *result)

Query if a given version of an extension is supported by an agent. All minor versions from 0 up to the returned `version_minor` must be supported.

Parameters

- **extension** – **[in]** Extension identifier.
- **agent** – **[in]** Agent.
- **version_major** – **[in]** Major version number.
- **version_minor** – **[out]** Minor version number.
- **result** – **[out]** Pointer to a memory location where the HSA runtime stores the result of the check. The result is true if the specified version of the extension is supported, and false otherwise. The result must be false if *hsa_system_extension_supported* returns false for the same extension version.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.

- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `extension` is not a valid extension, or `version_minor` is NULL, or `result` is NULL.

struct **hsa_agent_s**

#include <*hsa.h*> Struct containing an opaque handle to an agent, a device that participates in the HSA memory model. An agent can submit AQL packets for execution, and may also accept AQL packets for execution (agent dispatch packets or kernel dispatch packets launching HSAIL-derived binaries).

struct **hsa_cache_s**

#include <*hsa.h*> Cache handle.

4.13 Profiling

typedef struct *hsa_amd_profiling_dispatch_time_s* **hsa_amd_profiling_dispatch_time_t**

Structure containing profiling dispatch time information.

Times are reported as ticks in the domain of the HSA system clock. The HSA system clock tick and frequency is obtained via hsa_system_get_info.

typedef struct *hsa_amd_profiling_async_copy_time_s* **hsa_amd_profiling_async_copy_time_t**

Structure containing profiling async copy time information.

Times are reported as ticks in the domain of the HSA system clock. The HSA system clock tick and frequency is obtained via hsa_system_get_info.

hsa_status_t HSA_API hsa_amd_profiling_set_profiler_enabled (hsa_queue_t *queue, int enable)

Enable or disable profiling capability of a queue.

Parameters

- **queue** – **[in]** A valid queue.
- **enable** – **[in]** 1 to enable profiling. 0 to disable profiling.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_QUEUE** – The queue is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – queue is NULL.

hsa_status_t HSA_API hsa_amd_profiling_async_copy_enable (bool enable)

Enable or disable asynchronous memory copy profiling.

The runtime will provide the copy processing start timestamp and completion timestamp of each call to hsa_amd_memory_async_copy if the async copy profiling is enabled prior to the call to hsa_amd_memory_async_copy. The completion signal object is used to hold the last async copy start and end timestamp. The client can retrieve these timestamps via call to hsa_amd_profiling_get_async_copy_time.

Parameters

enable – **[in]** True to enable profiling. False to disable profiling.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_OUT_OF_RESOURCES** – Failed on allocating resources needed to profile the asynchronous copy.

hsa_status_t HSA_API hsa_amd_profiling_get_dispatch_time (hsa_agent_t agent, hsa_signal_t signal, hsa_amd_profiling_dispatch_time_t *time)

Retrieve packet processing time stamps.

Parameters

- **agent** – **[in]** The agent with which the signal was last used. For instance, if the profiled dispatch packet is dispatched onto queue Q, which was created on agent A, then this parameter must be A.
- **signal** – **[in]** A signal used as the completion signal of the dispatch packet to retrieve time stamps from. This dispatch packet must have been issued to a queue with profiling enabled and have already completed. Also the signal must not have yet been used in any other packet following the completion of the profiled dispatch packet.
- **time** – **[out]** Packet processing timestamps in the HSA system clock domain.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_SIGNAL** – The signal is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `time` is NULL.

hsa_status_t HSA_API hsa_amd_profiling_get_async_copy_time (hsa_signal_t signal, hsa_amd_profiling_async_copy_time_t *time)

Retrieve asynchronous copy timestamps.

Async copy profiling is enabled via call to hsa_amd_profiling_async_copy_enable.

Parameters

- **signal** – **[in]** A signal used as the completion signal of the call to hsa_amd_memory_async_copy.
- **time** – **[out]** Async copy processing timestamps in the HSA system clock domain.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_SIGNAL** – The signal is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `time` is NULL.

hsa_status_t HSA_API hsa_amd_profiling_convert_tick_to_system_domain (hsa_agent_t agent, uint64_t agent_tick, uint64_t *system_tick)

Computes the frequency ratio and offset between the agent clock and HSA system clock and converts the agent's tick to HSA system domain tick.

Parameters

- **agent** – **[in]** The agent used to retrieve the agent_tick. It is user's responsibility to make sure the tick number is from this agent, otherwise, the behavior is undefined.
- **agent_tick** – **[in]** The tick count retrieved from the specified `agent`.
- **system_tick** – **[out]** The translated HSA system domain clock counter tick.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_AGENT** – The agent is invalid.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `system_tick` is NULL;

hsa_status_t **hsa_amd_spm_acquire**(*hsa_agent_t* preferred_agent)

Acquire Stream Performance Monitor on an agent.

Acquire exclusive use of SPM on `preferred_agent`. See hsa_amd_spm_set_dest_buffer to provide a destination buffer to KFD to start recording and retrieve this data.

Parameters

preferred_agent – **[in]** Agent on which to acquire SPM

hsa_status_t **hsa_amd_spm_release**(*hsa_agent_t* preferred_agent)

Release Stream Performance Monitor on an agent.

Release exclusive use of SPM on `preferred_agent`. This will stop KFD writing SPM data. If a destination buffer is set, then data in the destination buffer is available to user when this function returns.

Parameters

preferred_agent – **[in]** Agent on which to release SPM

hsa_status_t **hsa_amd_spm_set_dest_buffer**(*hsa_agent_t* preferred_agent, size_t size_in_bytes, uint32_t *timeout, uint32_t *size_copied, void *dest, bool *is_data_loss)

Set up the current destination user mode buffer for stream performance counter data. KFD will start writing SPM data into the destination buffer. KFD will continue to copy data into the current destination buffer until any of the following functions are called.

- hsa_amd_spm_release
- hsa_amd_spm_set_dest_buffer with dest set to NULL
- hsa_amd_spm_set_dest_buffer with dest set to a new buffer

if `timeout` is non-0, the call will wait for up to `timeout` ms for the previous buffer to be filled. If previous buffer to be filled before timeout, the `timeout` will be updated value with the time remaining. If the timeout is exceeded, the function copies any partial data available into the previous user buffer and returns success. User should not access destination data while KFD is copying data. If the previous destination buffer was full, then `is_data_loss` flag is set. `dest` is CPU accessible memory. It could be malloc'ed memory or host allocated memory

Parameters

- **preferred_agent** – **[in]** Agent on which to set the dest buffer
- **size_in_bytes** – **[in]** size of the buffer
- **[in/out]** – timeout timeout in milliseconds
- **size_copied** – **[out]** number of bytes copied
- **dest** – **[in]** destination address. Set to NULL to stop copy on previous buffer
- **is_data_loss** – **[out]** true is data was lost

struct **hsa_amd_profiling_dispatch_time_s**

#include <hsa_ext_amd.h> Structure containing profiling dispatch time information.

Times are reported as ticks in the domain of the HSA system clock. The HSA system clock tick and frequency is obtained via hsa_system_get_info.

struct **hsa_amd_profiling_async_copy_time_s**

#include <hsa_ext_amd.h> Structure containing profiling async copy time information.

Times are reported as ticks in the domain of the HSA system clock. The HSA system clock tick and frequency is obtained via hsa_system_get_info.

4.14 Error codes

enum **hsa_status_t**

Status codes.

Values:

enumerator **HSA_STATUS_SUCCESS**

The function has been executed successfully.

enumerator **HSA_STATUS_INFO_BREAK**

A traversal over a list of elements has been interrupted by the application before completing.

enumerator **HSA_STATUS_ERROR**

A generic error has occurred.

enumerator **HSA_STATUS_ERROR_INVALID_ARGUMENT**

One of the actual arguments does not meet a precondition stated in the documentation of the corresponding formal argument.

enumerator **HSA_STATUS_ERROR_INVALID_QUEUE_CREATION**

The requested queue creation is not valid.

enumerator **HSA_STATUS_ERROR_INVALID_ALLOCATION**

The requested allocation is not valid.

enumerator **HSA_STATUS_ERROR_INVALID_AGENT**

The agent is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_REGION**

The memory region is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_SIGNAL**

The signal is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_QUEUE**

The queue is invalid.

enumerator **HSA_STATUS_ERROR_OUT_OF_RESOURCES**

The HSA runtime failed to allocate the necessary resources. This error may also occur when the HSA runtime needs to spawn threads or create internal OS-specific events.

enumerator **HSA_STATUS_ERROR_INVALID_PACKET_FORMAT**

The AQL packet is malformed.

enumerator **HSA_STATUS_ERROR_RESOURCE_FREE**

An error has been detected while releasing a resource.

enumerator **HSA_STATUS_ERROR_NOT_INITIALIZED**

An API other than hsa_init has been invoked while the reference count of the HSA runtime is 0.

enumerator **HSA_STATUS_ERROR_REFCOUNT_OVERFLOW**

The maximum reference count for the object has been reached.

enumerator **HSA_STATUS_ERROR_INCOMPATIBLE_ARGUMENTS**

The arguments passed to a functions are not compatible.

enumerator **HSA_STATUS_ERROR_INVALID_INDEX**

The index is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_ISA**

The instruction set architecture is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_ISA_NAME**

The instruction set architecture name is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_CODE_OBJECT**

The code object is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_EXECUTABLE**

The executable is invalid.

enumerator **HSA_STATUS_ERROR_FROZEN_EXECUTABLE**
The executable is frozen.

enumerator **HSA_STATUS_ERROR_INVALID_SYMBOL_NAME**
There is no symbol with the given name.

enumerator **HSA_STATUS_ERROR_VARIABLE_ALREADY_DEFINED**
The variable is already defined.

enumerator **HSA_STATUS_ERROR_VARIABLE_UNDEFINED**
The variable is undefined.

enumerator **HSA_STATUS_ERROR_EXCEPTION**
An HSAIL operation resulted in a hardware exception.

enumerator **HSA_STATUS_ERROR_INVALID_CODE_SYMBOL**
The code object symbol is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_EXECUTABLE_SYMBOL**
The executable symbol is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_FILE**
The file descriptor is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_CODE_OBJECT_READER**
The code object reader is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_CACHE**
The cache is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_WAVEFRONT**
The wavefront is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_SIGNAL_GROUP**
The signal group is invalid.

enumerator **HSA_STATUS_ERROR_INVALID_RUNTIME_STATE**
The HSA runtime is not in the configuration state.

enumerator **HSA_STATUS_ERROR_FATAL**
The queue received an error that may require process termination.

enum **[anonymous]**
Enumeration constants added to *hsa_status_t*.

Remark

Additions to hsa_status_t

Values:

enumerator **HSA_STATUS_ERROR_INVALID_MEMORY_POOL**

The memory pool is invalid.

enumerator **HSA_STATUS_ERROR_MEMORY_APERTURE_VIOLATION**

Agent accessed memory beyond the maximum legal address.

enumerator **HSA_STATUS_ERROR_ILLEGAL_INSTRUCTION**

Agent executed an invalid shader instruction.

enumerator **HSA_STATUS_ERROR_MEMORY_FAULT**

Agent attempted to access an inaccessible address. See hsa_amd_register_system_event_handler and HSA_AMD_GPU_MEMORY_FAULT_EVENT for more information on illegal accesses.

enumerator **HSA_STATUS_CU_MASK_REDUCED**

The CU mask was successfully set but the mask attempted to enable a CU which was disabled for the process. CUs disabled for the process remain disabled.

enumerator **HSA_STATUS_ERROR_OUT_OF_REGISTERS**

Exceeded number of VGPRs available on this agent

enumerator **HSA_STATUS_ERROR_RESOURCE_BUSY**

Resource is busy or temporarily unavailable

hsa_status_t HSA_API hsa_status_string (hsa_status_t status, const char **status_string)

Query additional information about a status code.

Parameters

- **status** – **[in]** Status code.
- **status_string** – **[out]** A NUL-terminated string that describes the error status.

Return values

- **HSA_STATUS_SUCCESS** – The function has been executed successfully.
- **HSA_STATUS_ERROR_NOT_INITIALIZED** – The HSA runtime has not been initialized.
- **HSA_STATUS_ERROR_INVALID_ARGUMENT** – `status` is an invalid status code, or `status_string` is NULL.

CHAPTER

FIVE

CONTRIBUTING TO ROCR

This document contains useful information required to contribute to ROCR.

5.1 Runtime design

ROCR consists of the following primary layers:

1. *C interface adaptors*
2. C++ interface classes and common functions
3. Device-specific implementations

The first layer provides interfaces to make ROCR APIs available to the user applications. The second and third layers comprise of the internal ROCR implementation, which is available for contribution.

Additionally, the runtime is dependent on a small utility library that provides simple common functions, limited operating system, compiler abstraction, and atomic operation interfaces.

The following sections list the important files present in the second and third layer.

5.2 C++ interface classes and common functions

The C++ interface layer provides abstract interface classes encapsulating commands to HSA signals, agents, and queues. This layer also contains the implementation of device-independent commands, such as `hsa_init`, `hsa_system_get_info`, and a default signal and queue implementation.

Files present in this layer:

- `runtime.h` (cpp)
- `agent.h`
- `queue.h`
- `signal.h`
- `memory_region.h` (cpp)
- `checked.h`
- `memory_database.h` (cpp)
- `default_signal.h` (cpp)

5.3 Device-specific implementations

The device-specific layer contains implementations of the C++ interface classes that implement HSA functionality for ROCm supported devices.

Files present in this layer:

- `amd_cpu_agent.h` (cpp)
- `amd_gpu_agent.h` (cpp)
- `amd_hw_aql_command_processor.h` (cpp)
- `amd_memory_region.h` (cpp)
- `amd_memory_registration.h` (cpp)
- `amd_topology.h` (cpp)
- `host_queue.h` (cpp)
- `interrupt_signal.h` (cpp)
- `hsa_ext_private_amd.h` (cpp)

5.4 Source and include directories

- `core`: Source code for AMD's implementation of the core HSA Runtime API's
- `cmake_modules`: CMake support modules and files
- `inc`: Public and AMD-specific header files exposing the HSA Runtime`s interfaces
- `libamdhsacode`: Code object definitions and interfaces
- `loader`: Loads code objects
- `utils`: Utilities required to build the core runtime

CHAPTER

SIX

LICENSE

INDEX

Symbols

H