ROCmSMI

Generated by Doxygen 1.8.18

Chapter 1

ROCm System Management Interface (ROCm SMI) Library

The ROCm System Management Interface Library, or ROCm SMI library, is part of the Radeon Open Compute ROCm software stack . It is a C library for Linux that provides a user space interface for applications to monitor and control GPU applications.

1.1 Important note about Versioning and Backward Compatibility

The ROCm SMI library is currently under development, and therefore subject to change either at the ABI or API level. The intention is to keep the API as stable as possible even while in development, but in some cases we may need to break backwards compatibility in order to ensure future stability and usability. Following Semantic Versioning rules, while the ROCm SMI library is in high state of change, the major version will remain 0, and backward compatibility is not ensured.

Once new development has leveled off, the major version will become greater than 0, and backward compatibility will be enforced between major versions.

1.2 Building ROCm SMI

1.2.0.0.1 Additional Required software for building In order to build the ROCm SMI library, the following components are required. Note that the software versions listed are what was used in development. Earlier versions are not guaranteed to work:

- CMake (v3.5.0)
- g++ (5.4.0)

In order to build the latest documentation, the following are required:

- DOxygen (1.8.11)
- latex (pdfTeX 3.14159265-2.6-1.40.16)

The source code for ROCm SMI is available on Github.

After the ROCm SMI library git repository has been cloned to a local Linux machine, building the library is achieved by following the typical CMake build sequence. Specifically,

```
$ mkdir -p build

$ cd build

$ cmake <location of root of ROCm SMI library CMakeLists.txt>

$ make

# Install library file and header; default location is /opt/rocm
```

$ make install The built library will appear in the `build` folder.

To build the rpm and deb packages follow the above steps with:

```
$ make package
```

1.2.0.0.2 Documentation The reference manual, `refman.pdf` will be in the `latex` directory upon a successful build.

1.2.0.0.3 Building the Tests In order to verify the build and capability of ROCm SMI on your system and to see an example of how ROCm SMI can be used, you may build and run the tests that are available in the repo. To build the tests, follow these steps:

```
# Set environment variables used in CMakeLists.txt file

$ ROCM_DIR=<parent dir.  to lib/ and inc/, containing RSMI library and header>

$ mkdir <location for test build>

$ cd <location for test build>

$ cmake -DROCM_DIR=$ROCM_DIR <ROCm SMI source root>/tests/rocm_smi_test
```

$ make To run the test, execute the program `rsmitst` that is built from the steps above.

1.3 Usage Basics

1.3.1 Device Indices

Many of the functions in the library take a "device index". The device index is a number greater than or equal to 0, and less than the number of devices detected, as determined by `rsmi_num_monitor_devices()`. The index is used to distinguish the detected devices from one another. It is important to note that a device may end up with a different index after a reboot, so an index should not be relied upon to be constant over reboots.

1.4 Hello ROCm SMI

The only required ROCm-SMI call for any program that wants to use ROCm-SMI is the `rsmi_init()` call. This call initializes some internal data structures that will be used by subsequent ROCm-SMI calls.

When ROCm-SMI is no longer being used, `rsmi_shut_down()` should be called. This provides a way to do any releasing of resources that ROCm-SMI may have held. In many cases, this may have no effect, but may be necessary in future versions of the library.

A simple "Hello World" type program that displays the device ID of detected devices would look like this:

```
#include <stdint.h>
#include "rocm_smi/rocm_smi.h"
int main() {
  rsmi_status_t ret;
  uint32_t num_devices;
  uint16_t dev_id;
  // We will skip return code checks for this example, but it
  // is recommended to always check this as some calls may not
  // apply for some devices or ROCm releases
  ret = rsmi_init(0);
  ret = rsmi_num_monitor_devices(&num_devices);
  for (int i=0; i < num_devices; ++i) {
    ret = rsmi_dev_id_get(i, &dev_id);
    // dev_id holds the device ID of device i, upon a
    // successful call
  }
  ret = rsmi_shut_down();
  return 0;
}
```

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Module Documentation

5.1 Initialization and Shutdown

These functions are used for initialization of ROCm SMI and clean up when done.

Functions

- rsmi_status_t rsmi_init (uint64_t init_flags)

  Initialize ROCm SMI.
- rsmi_status_t rsmi_shut_down (void)

  Shutdown ROCm SMI.

5.1.1 Detailed Description

These functions are used for initialization of ROCm SMI and clean up when done.

5.1.2 Function Documentation

5.1.2.1 rsmi_init()

```
rsmi_status_t rsmi_init (
            uint64_t init_flags )
```

Initialize ROCm SMI.

When called, this initializes internal data structures, including those corresponding to sources of information that SMI provides.

Parameters

in	*init_flags*	Bit flags that tell SMI how to initialze. Values of rsmi_init_flags_t may be OR'd together and passed through `init_flags` to modify how RSMI initializes.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

5.1.2.2 rsmi_shut_down()

```
rsmi_status_t rsmi_shut_down (
              void  )
```

Shutdown ROCm SMI.

Do any necessary clean up.

5.2 Identifier Queries

These functions provide identification information.

Functions

- rsmi_status_t rsmi_num_monitor_devices (uint32_t *num_devices)
  Get the number of devices that have monitor information.
- rsmi_status_t rsmi_dev_id_get (uint32_t dv_ind, uint16_t *id)
  Get the device id associated with the device with provided device index.
- rsmi_status_t rsmi_dev_vendor_id_get (uint32_t dv_ind, uint16_t *id)
  Get the device vendor id associated with the device with provided device index.
- rsmi_status_t rsmi_dev_name_get (uint32_t dv_ind, char *name, size_t len)
  Get the name string of a gpu device.
- rsmi_status_t rsmi_dev_brand_get (uint32_t dv_ind, char *brand, uint32_t len)
  Get the brand string of a gpu device.
- rsmi_status_t rsmi_dev_vendor_name_get (uint32_t dv_ind, char *name, size_t len)
  Get the name string for a give vendor ID.
- rsmi_status_t rsmi_dev_vram_vendor_get (uint32_t dv_ind, char *brand, uint32_t len)
  Get the vram vendor string of a gpu device.
- rsmi_status_t rsmi_dev_serial_number_get (uint32_t dv_ind, char *serial_num, uint32_t len)
  Get the serial number string for a device.
- rsmi_status_t rsmi_dev_subsystem_id_get (uint32_t dv_ind, uint16_t *id)
  Get the subsystem device id associated with the device with provided device index.
- rsmi_status_t rsmi_dev_subsystem_name_get (uint32_t dv_ind, char *name, size_t len)
  Get the name string for the device subsytem.
- rsmi_status_t rsmi_dev_drm_render_minor_get (uint32_t dv_ind, uint32_t *minor)
  Get the drm minor number associated with this device.
- rsmi_status_t rsmi_dev_subsystem_vendor_id_get (uint32_t dv_ind, uint16_t *id)
  Get the device subsystem vendor id associated with the device with provided device index.
- rsmi_status_t rsmi_dev_unique_id_get (uint32_t dv_ind, uint64_t *id)
  Get Unique ID.

5.2.1 Detailed Description

These functions provide identification information.

5.2.2 Function Documentation

5.2.2.1 rsmi_num_monitor_devices()

```
rsmi_status_t rsmi_num_monitor_devices (
            uint32_t * num_devices )
```

Get the number of devices that have monitor information.

The number of devices which have monitors is returned. Monitors are referenced by the index which can be between 0 and `num_devices` - 1.

Parameters

in,out	*num_devices*	Caller provided pointer to uint32_t. Upon successful call, the value num_devices will contain the number of monitor devices.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

5.2.2.2 rsmi_dev_id_get()

```
rsmi_status_t rsmi_dev_id_get (
            uint32_t dv_ind,
            uint16_t * id )
```

Get the device id associated with the device with provided device index.

Given a device index `dv_ind` and a pointer to a uint32_t `id`, this function will write the device id value to the uint64_t pointed to by `id`. This ID is an identification of the type of device, so calling this function for different devices will give the same value if they are kind of device. Consequently, this function should not be used to distinguish one device from another. rsmi_dev_pci_id_get() should be used to get a unique identifier.

Parameters

in	*dv_ind*	a device index
in,out	*id*	a pointer to uint64_t to which the device id will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.2.2.3 rsmi_dev_vendor_id_get()

```
rsmi_status_t rsmi_dev_vendor_id_get (
            uint32_t dv_ind,
            uint16_t * id )
```

Get the device vendor id associated with the device with provided device index.

Given a device index `dv_ind` and a pointer to a uint32_t `id`, this function will write the device vendor id value to the uint64_t pointed to by `id`.

Parameters

in	*dv_ind*	a device index
in,out	*id*	a pointer to uint64_t to which the device vendor id will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.2.2.4 rsmi_dev_name_get()

```
rsmi_status_t rsmi_dev_name_get (
            uint32_t dv_ind,
            char * name,
            size_t len )
```

Get the name string of a gpu device.

Given a device index `dv_ind`, a pointer to a caller provided char buffer `name`, and a length of this buffer `len`, this function will write the name of the device (up to `len` characters) to the buffer `name`.

If the integer ID associated with the device is not found in one of the system files containing device name information (e.g. /usr/share/misc/pci.ids), then this function will return the hex device ID as a string. Updating the system name files can be accompplished with "sudo update-pciids".

Parameters

in	*dv_ind*	a device index
in,out	*name*	a pointer to a caller provided char buffer to which the name will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.
in	*len*	the length of the caller provided buffer `name`.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if `len` bytes is not large enough to hold the entire name. In this case, only `len` bytes will be written.

5.2.2.5 rsmi_dev_brand_get()

```
rsmi_status_t rsmi_dev_brand_get (
            uint32_t dv_ind,
            char * brand,
            uint32_t len )
```

Get the brand string of a gpu device.

Given a device index `dv_ind`, a pointer to a caller provided char buffer `brand`, and a length of this buffer `len`, this function will write the brand of the device (up to `len` characters) to the buffer `brand`.

If the sku associated with the device is not found as one of the values contained within rsmi_dev_brand_get, then this function will return the device marketing name as a string instead of the brand name.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*brand*	a pointer to a caller provided char buffer to which the brand will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.
`in`	*len*	the length of the caller provided buffer `brand`.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if `len` bytes is not large enough to hold the entire name. In this case, only `len` bytes will be written.

5.2.2.6 rsmi_dev_vendor_name_get()

```
rsmi_status_t rsmi_dev_vendor_name_get (
            uint32_t dv_ind,
            char * name,
            size_t len )
```

Get the name string for a give vendor ID.

Given a device index `dv_ind`, a pointer to a caller provided char buffer `name`, and a length of this buffer `len`, this function will write the name of the vendor (up to `len` characters) buffer `name`. The `id` may be a device vendor or subsystem vendor ID.

If the integer ID associated with the vendor is not found in one of the system files containing device name information (e.g. /usr/share/misc/pci.ids), then this function will return the hex vendor ID as a string. Updating the system name files can be accompplished with "sudo update-pciids".

Parameters

in	*dv_ind*	a device index
in,out	*name*	a pointer to a caller provided char buffer to which the name will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.
in	*len*	the length of the caller provided buffer `name`.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if `len` bytes is not large enough to hold the entire name. In this case, only `len` bytes will be written.

5.2.2.7 rsmi_dev_vram_vendor_get()

```
rsmi_status_t rsmi_dev_vram_vendor_get (
            uint32_t dv_ind,
            char * brand,
            uint32_t len )
```

Get the vram vendor string of a gpu device.

Given a device index `dv_ind`, a pointer to a caller provided char buffer `brand`, and a length of this buffer `len`, this function will write the vram vendor of the device (up to `len` characters) to the buffer `brand`.

If the vram vendor for the device is not found as one of the values contained within rsmi_dev_vram_vendor_get, then this function will return the string 'unknown' instead of the vram vendor.

Parameters

in	*dv_ind*	a device index
in,out	*brand*	a pointer to a caller provided char buffer to which the vram vendor will be written
in	*len*	the length of the caller provided buffer `brand`.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

5.2.2.8 rsmi_dev_serial_number_get()

```
rsmi_status_t rsmi_dev_serial_number_get (
```

```
            uint32_t dv_ind,
            char * serial_num,
            uint32_t len )
```

Get the serial number string for a device.

Given a device index dv_ind, a pointer to a buffer of chars serial_num, and the length of the provided buffer len, this function will write the serial number string (up to len characters) to the buffer pointed to by serial_num.

Parameters

in	*dv_ind*	a device index
in,out	*serial_num*	a pointer to caller-provided memory to which the serial number will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.
in	*len*	the length of the caller provided buffer serial_num.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if len bytes is not large enough to hold the entire name. In this case, only len bytes will be written.

5.2.2.9 rsmi_dev_subsystem_id_get()

```
rsmi_status_t rsmi_dev_subsystem_id_get (
            uint32_t dv_ind,
            uint16_t * id )
```

Get the subsystem device id associated with the device with provided device index.

Given a device index dv_ind and a pointer to a uint32_t id, this function will write the subsystem device id value to the uint64_t pointed to by id.

Parameters

in	*dv_ind*	a device index
in,out	*id*	a pointer to uint64_t to which the subsystem device id will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.2.2.10 rsmi_dev_subsystem_name_get()

```
rsmi_status_t rsmi_dev_subsystem_name_get (
            uint32_t dv_ind,
            char * name,
            size_t len )
```

Get the name string for the device subsytem.

Given a device index `dv_ind`, a pointer to a caller provided char buffer `name`, and a length of this buffer `len`, this function will write the name of the device subsystem (up to `len` characters) to the buffer `name`.

If the integer ID associated with the sub-system is not found in one of the system files containing device name information (e.g. /usr/share/misc/pci.ids), then this function will return the hex sub-system ID as a string. Updating the system name files can be accompplished with "sudo update-pciids".

Parameters

`in`	*dv_ind*	a device index
`in,out`	*name*	a pointer to a caller provided char buffer to which the name will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.
`in`	*len*	the length of the caller provided buffer `name`.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if `len` bytes is not large enough to hold the entire name. In this case, only `len` bytes will be written.

5.2.2.11 rsmi_dev_drm_render_minor_get()

```
rsmi_status_t rsmi_dev_drm_render_minor_get (
            uint32_t dv_ind,
            uint32_t * minor )
```

Get the drm minor number associated with this device.

Given a device index `dv_ind`, find its render device file /dev/dri/renderDN where N corresponds to its minor number.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*minor*	a pointer to a uint32_t into which minor number will be copied

Return values

:	RSMI_STATUS_SUCCESS is returned upon successful call.
:	RSMI_STATUS_INIT_ERROR if failed to get minor number during initialization.
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.2.2.12 rsmi_dev_subsystem_vendor_id_get()

```
rsmi_status_t rsmi_dev_subsystem_vendor_id_get (
            uint32_t dv_ind,
            uint16_t * id )
```

Get the device subsystem vendor id associated with the device with provided device index.

Given a device index `dv_ind` and a pointer to a uint32_t `id`, this function will write the device subsystem vendor id value to the uint64_t pointed to by `id`.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*id*	a pointer to uint64_t to which the device subsystem vendor id will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.2.2.13 rsmi_dev_unique_id_get()

```
rsmi_status_t rsmi_dev_unique_id_get (
```

```
uint32_t dv_ind,
uint64_t * id )
```

Get Unique ID.

Given a device index `dv_ind` and a pointer to a uint64_t `id`, this function will write the unique ID of the GPU pointed to `id`.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*id*	a pointer to uint64_t to which the unique ID of the GPU is written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.3 PCIe Queries

These functions provide information about PCIe.

Functions

- rsmi_status_t rsmi_dev_pci_bandwidth_get (uint32_t dv_ind, rsmi_pcie_bandwidth_t *bandwidth)

  Get the list of possible PCIe bandwidths that are available.
- rsmi_status_t rsmi_dev_pci_id_get (uint32_t dv_ind, uint64_t *bdfid)

  Get the unique PCI device identifier associated for a device.
- rsmi_status_t rsmi_topo_numa_affinity_get (uint32_t dv_ind, uint32_t *numa_node)

  Get the NUMA node associated with a device.
- rsmi_status_t rsmi_dev_pci_throughput_get (uint32_t dv_ind, uint64_t *sent, uint64_t *received, uint64_t *max_pkt_sz)

  Get PCIe traffic information.
- rsmi_status_t rsmi_dev_pci_replay_counter_get (uint32_t dv_ind, uint64_t *counter)

  Get PCIe replay counter.

5.3.1 Detailed Description

These functions provide information about PCIe.

5.3.2 Function Documentation

5.3.2.1 rsmi_dev_pci_bandwidth_get()

```
rsmi_status_t rsmi_dev_pci_bandwidth_get (
            uint32_t dv_ind,
            rsmi_pcie_bandwidth_t * bandwidth )
```

Get the list of possible PCIe bandwidths that are available.

Given a device index `dv_ind` and a pointer to a to an rsmi_pcie_bandwidth_t structure `bandwidth`, this function will fill in `bandwidth` with the possible T/s values and associated number of lanes, and indication of the current selection.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*bandwidth*	a pointer to a caller provided rsmi_pcie_bandwidth_t structure to which the frequency information will be written

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

5.3.2.2 rsmi_dev_pci_id_get()

```
rsmi_status_t rsmi_dev_pci_id_get (
            uint32_t dv_ind,
            uint64_t * bdfid )
```

Get the unique PCI device identifier associated for a device.

Give a device index `dv_ind` and a pointer to a uint64_t `bdfid`, this function will write the Bus/Device/Function PCI identifier (BDFID) associated with device `dv_ind` to the value pointed to by `bdfid`.

The format of `bdfid` will be as follows:

```
BDFID = ((DOMAIN & 0xffffffff) << 32) | ((BUS & 0xff) << 8) |
                                ((DEVICE & 0x1f) <<3 ) | (FUNCTION & 0x7)
```

Name	Field
Domain	[64:32]
Reserved	[31:16]
Bus	[15: 8]
Device	[7: 3]
Function	[2: 0]

Parameters

`in`	*dv_ind*	a device index
`in,out`	*bdfid*	a pointer to uint64_t to which the device bdfid value will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.3.2.3 rsmi_topo_numa_affinity_get()

```
rsmi_status_t rsmi_topo_numa_affinity_get (
            uint32_t dv_ind,
            uint32_t * numa_node )
```

Get the NUMA node associated with a device.

Given a device index `dv_ind` and a pointer to a uint32_t `numa_node`, this function will retrieve the NUMA node value associated with device `dv_ind` and store the value at location pointed to by `numa_node`.

Parameters

in	*dv_ind*	a device index
in,out	*numa_node*	pointer to location where NUMA node value will be written. If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.3.2.4 rsmi_dev_pci_throughput_get()

```
rsmi_status_t rsmi_dev_pci_throughput_get (
            uint32_t dv_ind,
            uint64_t * sent,
            uint64_t * received,
            uint64_t * max_pkt_sz )
```

Get PCIe traffic information.

Give a device index `dv_ind` and pointers to a uint64_t's, `sent`, `received` and `max_pkt_sz`, this function will write the number of bytes sent and received in 1 second to `sent` and `received`, respectively. The maximum possible packet size will be written to `max_pkt_sz`.

Parameters

in	*dv_ind*	a device index
in,out	*sent*	a pointer to uint64_t to which the number of bytes sent will be written in 1 second. If pointer is NULL, it will be ignored.
in,out	*received*	a pointer to uint64_t to which the number of bytes received will be written. If pointer is NULL, it will be ignored.
in,out	*max_pkt_sz*	a pointer to uint64_t to which the maximum packet size will be written. If pointer is NULL, it will be ignored.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments

5.3.2.5 rsmi_dev_pci_replay_counter_get()

```
rsmi_status_t rsmi_dev_pci_replay_counter_get (
            uint32_t dv_ind,
            uint64_t * counter )
```

Get PCIe replay counter.

Given a device index `dv_ind` and a pointer to a uint64_t `counter`, this function will write the sum of the number of NAK's received by the GPU and the NAK's generated by the GPU to memory pointed to by `counter`.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*counter*	a pointer to uint64_t to which the sum of the NAK's received and generated by the GPU is written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.4 PCIe Control

These functions provide some control over PCIe.

Functions

- rsmi_status_t rsmi_dev_pci_bandwidth_set (uint32_t dv_ind, uint64_t bw_bitmask)

  Control the set of allowed PCIe bandwidths that can be used.

5.4.1 Detailed Description

These functions provide some control over PCIe.

5.4.2 Function Documentation

5.4.2.1 rsmi_dev_pci_bandwidth_set()

```
rsmi_status_t rsmi_dev_pci_bandwidth_set (
            uint32_t dv_ind,
            uint64_t bw_bitmask )
```

Control the set of allowed PCIe bandwidths that can be used.

Given a device index `dv_ind` and a 64 bit bitmask `bw_bitmask`, this function will limit the set of allowable bandwidths. If a bit in `bw_bitmask` has a value of 1, then the frequency (as ordered in an rsmi_frequencies_t returned by rsmi_dev_gpu_clk_freq_get()) corresponding to that bit index will be allowed.

This function will change the performance level to RSMI_DEV_PERF_LEVEL_MANUAL in order to modify the set of allowable band_widths. Caller will need to set to RSMI_DEV_PERF_LEVEL_AUTO in order to get back to default state.

All bits with indices greater than or equal to the value of the rsmi_frequencies_t::num_supported field of rsmi_pcie_bandwidth_t will be ignored.

Parameters

in	*dv_ind*	a device index
in	*bw_bitmask*	A bitmask indicating the indices of the bandwidths that are to be enabled (1) and disabled (0). Only the lowest rsmi_frequencies_t::num_supported (of rsmi_pcie_bandwidth_t) bits of this mask are relevant.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_PERMISSION	function requires root access

5.5 Power Queries

These functions provide information about power usage.

Functions

- rsmi_status_t rsmi_dev_power_ave_get (uint32_t dv_ind, uint32_t sensor_ind, uint64_t *power)

  Get the average power consumption of the device with provided device index.
- rsmi_status_t rsmi_dev_power_cap_get (uint32_t dv_ind, uint32_t sensor_ind, uint64_t *cap)

  Get the cap on power which, when reached, causes the system to take action to reduce power.
- rsmi_status_t rsmi_dev_power_cap_range_get (uint32_t dv_ind, uint32_t sensor_ind, uint64_t *max, uint64_t *min)

  Get the range of valid values for the power cap.

5.5.1 Detailed Description

These functions provide information about power usage.

5.5.2 Function Documentation

5.5.2.1 rsmi_dev_power_ave_get()

```
rsmi_status_t rsmi_dev_power_ave_get (
            uint32_t dv_ind,
            uint32_t sensor_ind,
            uint64_t * power )
```

Get the average power consumption of the device with provided device index.

Given a device index `dv_ind` and a pointer to a uint64_t `power`, this function will write the current average power consumption (in microwatts) to the uint64_t pointed to by `power`.

Parameters

in	*dv_ind*	a device index
in	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
in,out	*power*	a pointer to uint64_t to which the average power consumption will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.5.2.2 rsmi_dev_power_cap_get()

```
rsmi_status_t rsmi_dev_power_cap_get (
            uint32_t dv_ind,
            uint32_t sensor_ind,
            uint64_t * cap )
```

Get the cap on power which, when reached, causes the system to take action to reduce power.

When power use rises above the value `power`, the system will take action to reduce power use. The power level returned through `power` will be in microWatts.

Parameters

`in`	*dv_ind*	a device index
`in`	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
`in,out`	*cap*	a pointer to a uint64_t that indicates the power cap, in microwatts If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.5.2.3 rsmi_dev_power_cap_range_get()

```
rsmi_status_t rsmi_dev_power_cap_range_get (
            uint32_t dv_ind,
            uint32_t sensor_ind,
            uint64_t * max,
            uint64_t * min )
```

Get the range of valid values for the power cap.

This function will return the maximum possible valid power cap `max` and the minimum possible valid power cap `min`

Parameters

`in`	*dv_ind*	a device index
`in`	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
`in,out`	*max*	a pointer to a uint64_t that indicates the maximum possible power cap, in microwatts If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.
`in,out`	*min*	a pointer to a uint64_t that indicates the minimum possible power cap, in microwatts If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.6 Power Control

These functions provide ways to control power usage.

Functions

- rsmi_status_t rsmi_dev_power_cap_set (uint32_t dv_ind, uint32_t sensor_ind, uint64_t cap)

  Set the power cap value.
- rsmi_status_t rsmi_dev_power_profile_set (uint32_t dv_ind, uint32_t reserved, rsmi_power_profile_preset_masks_t profile)

  Set the power profile.

5.6.1 Detailed Description

These functions provide ways to control power usage.

5.6.2 Function Documentation

5.6.2.1 rsmi_dev_power_cap_set()

```
rsmi_status_t rsmi_dev_power_cap_set (
            uint32_t dv_ind,
            uint32_t sensor_ind,
            uint64_t cap )
```

Set the power cap value.

This function will set the power cap to the provided value `cap`. `cap` must be between the minimum and maximum power cap values set by the system, which can be obtained from rsmi_dev_power_cap_range_get.

Parameters

in	*dv_ind*	a device index
in	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
in	*cap*	a uint64_t that indicates the desired power cap, in microwatts

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_PERMISSION	function requires root access

5.6.2.2 rsmi_dev_power_profile_set()

```
rsmi_status_t rsmi_dev_power_profile_set (
            uint32_t dv_ind,
            uint32_t reserved,
            rsmi_power_profile_preset_masks_t profile )
```

Set the power profile.

Given a device index `dv_ind` and a `profile`, this function will attempt to set the current profile to the provided profile. The provided profile must be one of the currently supported profiles, as indicated by a call to rsmi_dev_power_profile_presets_get()

Parameters

in	*dv_ind*	a device index
in	*reserved*	Not currently used. Set to 0.
in	*profile*	a rsmi_power_profile_preset_masks_t that hold the mask of the desired new power profile

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.
RSMI_STATUS_PERMISSION	function requires root access

5.7 Memory Queries

These functions provide information about memory systems.

Functions

- rsmi_status_t rsmi_dev_memory_total_get (uint32_t dv_ind, rsmi_memory_type_t mem_type, uint64_t *total)

  Get the total amount of memory that exists.
- rsmi_status_t rsmi_dev_memory_usage_get (uint32_t dv_ind, rsmi_memory_type_t mem_type, uint64_t *used)

  Get the current memory usage.
- rsmi_status_t rsmi_dev_memory_busy_percent_get (uint32_t dv_ind, uint32_t *busy_percent)

  Get percentage of time any device memory is being used.
- rsmi_status_t rsmi_dev_memory_reserved_pages_get (uint32_t dv_ind, uint32_t *num_pages, rsmi_retired_page_record_t *records)

  Get information about reserved ("retired") memory pages.

5.7.1 Detailed Description

These functions provide information about memory systems.

5.7.2 Function Documentation

5.7.2.1 rsmi_dev_memory_total_get()

```
rsmi_status_t rsmi_dev_memory_total_get (
            uint32_t dv_ind,
            rsmi_memory_type_t mem_type,
            uint64_t * total )
```

Get the total amount of memory that exists.

Given a device index `dv_ind`, a type of memory `mem_type`, and a pointer to a uint64_t `total`, this function will write the total amount of `mem_type` memory that exists to the location pointed to by `total`.

Parameters

`in`	*dv_ind*	a device index
`in`	*mem_type*	The type of memory for which the total amount will be found
`in,out`	*total*	a pointer to uint64_t to which the total amount of memory will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.7.2.2 rsmi_dev_memory_usage_get()

```
rsmi_status_t rsmi_dev_memory_usage_get (
            uint32_t dv_ind,
            rsmi_memory_type_t mem_type,
            uint64_t * used )
```

Get the current memory usage.

Given a device index `dv_ind`, a type of memory `mem_type`, and a pointer to a uint64_t `usage`, this function will write the amount of `mem_type` memory that that is currently being used to the location pointed to by `used`.

Parameters

`in`	*dv_ind*	a device index
`in`	*mem_type*	The type of memory for which the amount being used will be found
`in,out`	*used*	a pointer to uint64_t to which the amount of memory currently being used will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.7.2.3 rsmi_dev_memory_busy_percent_get()

```
rsmi_status_t rsmi_dev_memory_busy_percent_get (
            uint32_t dv_ind,
            uint32_t * busy_percent )
```

Get percentage of time any device memory is being used.

Given a device index `dv_ind`, this function returns the percentage of time that any device memory is being used for the specified device.

Parameters

in	*dv_ind*	a device index
in,out	*busy_percent*	a pointer to the uint32_t to which the busy percent will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.7.2.4 rsmi_dev_memory_reserved_pages_get()

```
rsmi_status_t rsmi_dev_memory_reserved_pages_get (
            uint32_t dv_ind,
            uint32_t * num_pages,
            rsmi_retired_page_record_t * records )
```

Get information about reserved ("retired") memory pages.

Given a device index `dv_ind`, this function returns retired page information `records` corresponding to the device with the provided device index `dv_ind`. The number of retired page records is returned through `num_pages`. `records` may be NULL on input. In this case, the number of records available for retrieval will be returned through `num_pages`.

Parameters

in	*dv_ind*	a device index
in,out	*num_pages*	a pointer to a uint32. As input, the value passed through this parameter is the number of rsmi_retired_page_record_t's that may be safely written to the memory pointed to by `records`. This is the limit on how many records will be written to `records`. On return, `num_pages` will contain the number of records written to `records`, or the number of records that could have been written if enough memory had been provided. If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.
in,out	*records*	A pointer to a block of memory to which the rsmi_retired_page_record_t values will be written. This value may be NULL. In this case, this function can be used to query how many records are available to read.

Return values

RSMI_STATUS_SUCCESS	call was successful

Return values

RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if more records were available than allowed by the provided, allocated memory.

5.8 Physical State Queries

These functions provide information about the physical characteristics of the device.

Functions

- rsmi_status_t rsmi_dev_fan_rpms_get (uint32_t dv_ind, uint32_t sensor_ind, int64_t ∗speed)

  Get the fan speed in RPMs of the device with the specified device index and 0-based sensor index.
- rsmi_status_t rsmi_dev_fan_speed_get (uint32_t dv_ind, uint32_t sensor_ind, int64_t ∗speed)

  Get the fan speed for the specified device as a value relative to RSMI_MAX_FAN_SPEED.
- rsmi_status_t rsmi_dev_fan_speed_max_get (uint32_t dv_ind, uint32_t sensor_ind, uint64_t ∗max_speed)

  Get the max. fan speed of the device with provided device index.
- rsmi_status_t rsmi_dev_temp_metric_get (uint32_t dv_ind, uint32_t sensor_type, rsmi_temperature_metric_t metric, int64_t ∗temperature)

  Get the temperature metric value for the specified metric, from the specified temperature sensor on the specified device.

5.8.1 Detailed Description

These functions provide information about the physical characteristics of the device.

5.8.2 Function Documentation

5.8.2.1 rsmi_dev_fan_rpms_get()

```
rsmi_status_t rsmi_dev_fan_rpms_get (
            uint32_t dv_ind,
            uint32_t sensor_ind,
            int64_t * speed )
```

Get the fan speed in RPMs of the device with the specified device index and 0-based sensor index.

Given a device index `dv_ind` and a pointer to a uint32_t `speed`, this function will write the current fan speed in RPMs to the uint32_t pointed to by `speed`

Parameters

`in`	*dv_ind*	a device index
`in`	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
`in,out`	*speed*	a pointer to uint32_t to which the speed will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.8.2.2 rsmi_dev_fan_speed_get()

```
rsmi_status_t rsmi_dev_fan_speed_get (
            uint32_t dv_ind,
            uint32_t sensor_ind,
            int64_t * speed )
```

Get the fan speed for the specified device as a value relative to RSMI_MAX_FAN_SPEED.

Given a device index `dv_ind` and a pointer to a uint32_t `speed`, this function will write the current fan speed (a value between 0 and the maximum fan speed, RSMI_MAX_FAN_SPEED) to the uint32_t pointed to by `speed`

Parameters

`in`	*dv_ind*	a device index
`in`	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
`in,out`	*speed*	a pointer to uint32_t to which the speed will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.8.2.3 rsmi_dev_fan_speed_max_get()

```
rsmi_status_t rsmi_dev_fan_speed_max_get (
            uint32_t dv_ind,
            uint32_t sensor_ind,
            uint64_t * max_speed )
```

Get the max. fan speed of the device with provided device index.

Given a device index `dv_ind` and a pointer to a uint32_t `max_speed`, this function will write the maximum fan speed possible to the uint32_t pointed to by `max_speed`

Parameters

in	*dv_ind*	a device index
in	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
in,out	*max_speed*	a pointer to uint32_t to which the maximum speed will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.8.2.4 rsmi_dev_temp_metric_get()

```
rsmi_status_t rsmi_dev_temp_metric_get (
            uint32_t dv_ind,
            uint32_t sensor_type,
            rsmi_temperature_metric_t metric,
            int64_t * temperature )
```

Get the temperature metric value for the specified metric, from the specified temperature sensor on the specified device.

Given a device index `dv_ind`, a sensor type `sensor_type`, a rsmi_temperature_metric_t `metric` and a pointer to an int64_t `temperature`, this function will write the value of the metric indicated by `metric` and `sensor_type` to the memory location `temperature`.

Parameters

in	*dv_ind*	a device index
in	*sensor_type*	part of device from which temperature should be obtained. This should come from the enum rsmi_temperature_type_t
in	*metric*	enum indicated which temperature value should be retrieved
in,out	*temperature*	a pointer to int64_t to which the temperature will be written, in millidegrees Celcius. If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments

Return values

RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.9 Physical State Control

These functions provide control over the physical state of a device.

Functions

- rsmi_status_t rsmi_dev_fan_reset (uint32_t dv_ind, uint32_t sensor_ind)

  Reset the fan to automatic driver control.
- rsmi_status_t rsmi_dev_fan_speed_set (uint32_t dv_ind, uint32_t sensor_ind, uint64_t speed)

  Set the fan speed for the specified device with the provided speed, in RPMs.

5.9.1 Detailed Description

These functions provide control over the physical state of a device.

5.9.2 Function Documentation

5.9.2.1 rsmi_dev_fan_reset()

```
rsmi_status_t rsmi_dev_fan_reset (
            uint32_t dv_ind,
            uint32_t sensor_ind )
```

Reset the fan to automatic driver control.

This function returns control of the fan to the system

Parameters

in	*dv_ind*	a device index
in	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments

5.9.2.2 rsmi_dev_fan_speed_set()

```
rsmi_status_t rsmi_dev_fan_speed_set (
```

```
uint32_t dv_ind,
uint32_t sensor_ind,
uint64_t speed )
```

Set the fan speed for the specified device with the provided speed, in RPMs.

Given a device index `dv_ind` and a integer value indicating speed `speed`, this function will attempt to set the fan speed to `speed`. An error will be returned if the specified speed is outside the allowable range for the device. The maximum value is 255 and the minimum is 0.

Parameters

`in`	*dv_ind*	a device index
`in`	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
`in`	*speed*	the speed to which the function will attempt to set the fan

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_PERMISSION	function requires root access

5.10 Clock, Power and Performance Queries

These functions provide information about clock frequencies and performance.

Functions

- rsmi_status_t rsmi_dev_busy_percent_get (uint32_t dv_ind, uint32_t *busy_percent)

  Get percentage of time device is busy doing any processing.
- rsmi_status_t rsmi_dev_perf_level_get (uint32_t dv_ind, rsmi_dev_perf_level_t *perf)

  Get the performance level of the device with provided device index.
- rsmi_status_t rsmi_dev_overdrive_level_get (uint32_t dv_ind, uint32_t *od)

  Get the overdrive percent associated with the device with provided device index.
- rsmi_status_t rsmi_dev_gpu_clk_freq_get (uint32_t dv_ind, rsmi_clk_type_t clk_type, rsmi_frequencies_t *f)

  Get the list of possible system clock speeds of device for a specified clock type.
- rsmi_status_t rsmi_dev_od_volt_info_get (uint32_t dv_ind, rsmi_od_volt_freq_data_t *odv)

  This function retrieves the voltage/frequency curve information.
- rsmi_status_t rsmi_dev_od_volt_curve_regions_get (uint32_t dv_ind, uint32_t *num_regions, rsmi_freq_volt_region_t *buffer)

  This function will retrieve the current valid regions in the frequency/voltage space.
- rsmi_status_t rsmi_dev_power_profile_presets_get (uint32_t dv_ind, uint32_t sensor_ind, rsmi_power_profile_status_t *status)

  Get the list of available preset power profiles and an indication of which profile is currently active.

5.10.1 Detailed Description

These functions provide information about clock frequencies and performance.

5.10.2 Function Documentation

5.10.2.1 rsmi_dev_busy_percent_get()

```
rsmi_status_t rsmi_dev_busy_percent_get (
            uint32_t dv_ind,
            uint32_t * busy_percent )
```

Get percentage of time device is busy doing any processing.

Given a device index `dv_ind`, this function returns the percentage of time that the specified device is busy. The device is considered busy if any one or more of its sub-blocks are working, and idle if none of the sub-blocks are working.

Parameters

in	*dv_ind*	a device index
in,out	*busy_percent*	a pointer to the uint32_t to which the busy percent will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.10.2.2 rsmi_dev_perf_level_get()

```
rsmi_status_t rsmi_dev_perf_level_get (
            uint32_t dv_ind,
            rsmi_dev_perf_level_t * perf )
```

Get the performance level of the device with provided device index.

Given a device index `dv_ind` and a pointer to a uint32_t `perf`, this function will write the rsmi_dev_perf_level_t to the uint32_t pointed to by `perf`

Parameters

`in`	*dv_ind*	a device index
`in,out`	*perf*	a pointer to rsmi_dev_perf_level_t to which the performance level will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.10.2.3 rsmi_dev_overdrive_level_get()

```
rsmi_status_t rsmi_dev_overdrive_level_get (
            uint32_t dv_ind,
            uint32_t * od )
```

Get the overdrive percent associated with the device with provided device index.

Given a device index `dv_ind` and a pointer to a uint32_t `od`, this function will write the overdrive percentage to the uint32_t pointed to by `od`

Parameters

in	*dv_ind*	a device index
in,out	*od*	a pointer to uint32_t to which the overdrive percentage will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.10.2.4 rsmi_dev_gpu_clk_freq_get()

```
rsmi_status_t rsmi_dev_gpu_clk_freq_get (
            uint32_t dv_ind,
            rsmi_clk_type_t clk_type,
            rsmi_frequencies_t * f )
```

Get the list of possible system clock speeds of device for a specified clock type.

Given a device index `dv_ind`, a clock type `clk_type`, and a pointer to a to an rsmi_frequencies_t structure `f`, this function will fill in `f` with the possible clock speeds, and indication of the current clock speed selection.

Parameters

in	*dv_ind*	a device index
in	*clk_type*	the type of clock for which the frequency is desired
in,out	*f*	a pointer to a caller provided rsmi_frequencies_t structure to which the frequency information will be written. Frequency values are in Hz. If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.10.2.5 rsmi_dev_od_volt_info_get()

```
rsmi_status_t rsmi_dev_od_volt_info_get (
            uint32_t dv_ind,
            rsmi_od_volt_freq_data_t * odv )
```

This function retrieves the voltage/frequency curve information.

Given a device index `dv_ind` and a pointer to a rsmi_od_volt_freq_data_t structure `odv`, this function will populate `odv`. See rsmi_od_volt_freq_data_t for more details.

Parameters

in	*dv_ind*	a device index
in,out	*odv*	a pointer to an rsmi_od_volt_freq_data_t structure If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.10.2.6 rsmi_dev_od_volt_curve_regions_get()

```
rsmi_status_t rsmi_dev_od_volt_curve_regions_get (
            uint32_t dv_ind,
            uint32_t * num_regions,
            rsmi_freq_volt_region_t * buffer )
```

This function will retrieve the current valid regions in the frequency/voltage space.

Given a device index `dv_ind`, a pointer to an unsigned integer `num_regions` and a buffer of rsmi_freq_volt_region_t structures, `buffer`, this function will populate `buffer` with the current frequency-volt space regions. The caller should assign `buffer` to memory that can be written to by this function. The caller should also indicate the number of rsmi_freq_volt_region_t structures that can safely be written to `buffer` in `num_regions`.

The number of regions to expect this function provide (`num_regions`) can be obtained by calling rsmi_dev_od_volt_info_get().

Parameters

in	*dv_ind*	a device index
in,out	*num_regions*	As input, this is the number of rsmi_freq_volt_region_t structures that can be written to `buffer`. As output, this is the number of rsmi_freq_volt_region_t structures that were actually written. If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Parameters

in,out	*buffer*	a caller provided buffer to which rsmi_freq_volt_region_t structures will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.10.2.7 rsmi_dev_power_profile_presets_get()

```
rsmi_status_t rsmi_dev_power_profile_presets_get (
            uint32_t dv_ind,
            uint32_t sensor_ind,
            rsmi_power_profile_status_t * status )
```

Get the list of available preset power profiles and an indication of which profile is currently active.

Given a device index dv_ind and a pointer to a rsmi_power_profile_status_t status, this function will set the bits of the rsmi_power_profile_status_t.available_profiles bit field of status to 1 if the profile corresponding to the respective rsmi_power_profile_preset_masks_t profiles are enabled. For example, if both the VIDEO and VR power profiles are available selections, then RSMI_PWR_PROF_PRST_VIDEO_MASK AND'ed with rsmi_power_profile_status_t.available_profiles will be non-zero as will RSMI_PWR_PROF_PRST_VR_MASK AND'ed with rsmi_power_profile_status_t.available_profiles. Additionally, rsmi_power_profile_status_t.current will be set to the rsmi_power_profile_preset_masks_t of the profile that is currently active.

Parameters

in	*dv_ind*	a device index
in	*sensor_ind*	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
in,out	*status*	a pointer to rsmi_power_profile_status_t that will be populated by a call to this function If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.11 Clock, Power and Performance Control

These functions provide control over clock frequencies, power and performance.

Functions

- rsmi_status_t rsmi_dev_perf_level_set (int32_t dv_ind, rsmi_dev_perf_level_t perf_lvl)

  Set the PowerPlay performance level associated with the device with provided device index with the provided value.
- rsmi_status_t rsmi_dev_overdrive_level_set (int32_t dv_ind, uint32_t od)

  Set the overdrive percent associated with the device with provided device index with the provided value. See details for WARNING.
- rsmi_status_t rsmi_dev_gpu_clk_freq_set (uint32_t dv_ind, rsmi_clk_type_t clk_type, uint64_t freq_bitmask)

  Control the set of allowed frequencies that can be used for the specified clock.

5.11.1 Detailed Description

These functions provide control over clock frequencies, power and performance.

5.11.2 Function Documentation

5.11.2.1 rsmi_dev_perf_level_set()

```
rsmi_status_t rsmi_dev_perf_level_set (
            int32_t dv_ind,
            rsmi_dev_perf_level_t perf_lvl )
```

Set the PowerPlay performance level associated with the device with provided device index with the provided value.

Given a device index `dv_ind` and an rsmi_dev_perf_level_t `perf_level`, this function will set the PowerPlay performance level for the device to the value `perf_lvl`.

Parameters

in	*dv_ind*	a device index
in	*perf_lvl*	the value to which the performance level should be set

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_PERMISSION	function requires root access

5.11.2.2 rsmi_dev_overdrive_level_set()

```
rsmi_status_t rsmi_dev_overdrive_level_set (
            int32_t dv_ind,
            uint32_t od )
```

Set the overdrive percent associated with the device with provided device index with the provided value. See details for WARNING.

Given a device index `dv_ind` and an overdrive level `od`, this function will set the overdrive level for the device to the value `od`. The overdrive level is an integer value between 0 and 20, inclusive, which represents the overdrive percentage; e.g., a value of 5 specifies an overclocking of 5%.

The overdrive level is specific to the gpu system clock.

The overdrive level is the percentage above the maximum Performance Level to which overclocking will be limited. The overclocking percentage does not apply to clock speeds other than the maximum. This percentage is limited to 20%.

******WARNING****** Operating your AMD GPU outside of official AMD specifications or outside of factory settings, including but not limited to the conducting of overclocking (including use of this overclocking software, even if such software has been directly or indirectly provided by AMD or otherwise affiliated in any way with AMD), may cause damage to your AMD GPU, system components and/or result in system failure, as well as cause other problems. DAMAGES CAUSED BY USE OF YOUR AMD GPU OUTSIDE OF OFFICIAL AMD SPECIFICATIONS OR OUTSIDE OF FACTORY SETTINGS ARE NOT COVERED UNDER ANY AMD PRODUCT WARRANTY A↩ ND MAY NOT BE COVERED BY YOUR BOARD OR SYSTEM MANUFACTURER'S WARRANTY. Please use this utility with caution.

Parameters

in	*dv_ind*	a device index
in	*od*	the value to which the overdrive level should be set

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_PERMISSION	function requires root access

5.11.2.3 rsmi_dev_gpu_clk_freq_set()

```
rsmi_status_t rsmi_dev_gpu_clk_freq_set (
            uint32_t dv_ind,
            rsmi_clk_type_t clk_type,
            uint64_t freq_bitmask )
```

Control the set of allowed frequencies that can be used for the specified clock.

Given a device index `dv_ind`, a clock type `clk_type`, and a 64 bit bitmask `freq_bitmask`, this function will limit the set of allowable frequencies. If a bit in `freq_bitmask` has a value of 1, then the frequency (as ordered in an rsmi_frequencies_t returned by rsmi_dev_gpu_clk_freq_get()) corresponding to that bit index will be allowed.

This function will change the performance level to RSMI_DEV_PERF_LEVEL_MANUAL in order to modify the set of allowable frequencies. Caller will need to set to RSMI_DEV_PERF_LEVEL_AUTO in order to get back to default state.

All bits with indices greater than or equal to rsmi_frequencies_t::num_supported will be ignored.

Parameters

`in`	*dv_ind*	a device index
`in`	*clk_type*	the type of clock for which the set of frequencies will be modified
`in`	*freq_bitmask*	A bitmask indicating the indices of the frequencies that are to be enabled (1) and disabled (0). Only the lowest rsmi_frequencies_t.num_supported bits of this mask are relevant.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_PERMISSION	function requires root access

5.12 Version Queries

These functions provide version information about various subsystems.

Functions

- rsmi_status_t rsmi_version_get (rsmi_version_t *version)

  Get the build version information for the currently running build of RSMI.
- rsmi_status_t rsmi_version_str_get (rsmi_sw_component_t component, char *ver_str, uint32_t len)

  Get the driver version string for the current system.
- rsmi_status_t rsmi_dev_vbios_version_get (uint32_t dv_ind, char *vbios, uint32_t len)

  Get the VBIOS identifer string.
- rsmi_status_t rsmi_dev_firmware_version_get (uint32_t dv_ind, rsmi_fw_block_t block, uint64_t *fw_version)

  Get the firmware versions for a device.

5.12.1 Detailed Description

These functions provide version information about various subsystems.

5.12.2 Function Documentation

5.12.2.1 rsmi_version_get()

```
rsmi_status_t rsmi_version_get (
            rsmi_version_t * version )
```

Get the build version information for the currently running build of RSMI.

Get the major, minor, patch and build string for RSMI build currently in use through `version`

Parameters

`in,out`	*version*	A pointer to an rsmi_version_t structure that will be updated with the version information upon return.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call

5.12.2.2 rsmi_version_str_get()

```
rsmi_status_t rsmi_version_str_get (
```

```
rsmi_sw_component_t component,
char * ver_str,
uint32_t len )
```

Get the driver version string for the current system.

Given a software component `component`, a pointer to a char buffer, `ver_str`, this function will write the driver version string (up to `len` characters) for the current system to `ver_str`. The caller must ensure that it is safe to write at least `len` characters to `ver_str`.

Parameters

`in`	*component*	The component for which the version string is being requested
`in,out`	*ver_str*	A pointer to a buffer of char's to which the version of `component` will be written
`in`	*len*	the length of the caller provided buffer `name`.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if `len` bytes is not large enough to hold the entire name. In this case, only `len` bytes will be written.

5.12.2.3 rsmi_dev_vbios_version_get()

```
rsmi_status_t rsmi_dev_vbios_version_get (
uint32_t dv_ind,
char * vbios,
uint32_t len )
```

Get the VBIOS identifer string.

Given a device ID `dv_ind`, and a pointer to a char buffer, `vbios`, this function will write the VBIOS string (up to `len` characters) for device `dv_ind` to `vbios`. The caller must ensure that it is safe to write at least `len` characters to `vbios`.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*vbios*	A pointer to a buffer of char's to which the VBIOS name will be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.
`in`	*len*	The number of char's pointed to by `vbios` which can safely be written to by this function.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.12.2.4 rsmi_dev_firmware_version_get()

```
rsmi_status_t rsmi_dev_firmware_version_get (
            uint32_t dv_ind,
            rsmi_fw_block_t block,
            uint64_t * fw_version )
```

Get the firmware versions for a device.

Given a device ID `dv_ind`, and a pointer to a uint64_t, `fw_version`, this function will write the FW Versions as a string (up to `len` characters) for device `dv_ind` to `vbios`. The caller must ensure that it is safe to write at least `len` characters to `vbios`.

Parameters

`in`	*dv_ind*	a device index
`in`	*block*	The firmware block for which the version is being requested
`in,out`	*fw_version*	The version for the firmware block If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.13 Error Queries

These functions provide error information about RSMI calls as well as device errors.

Functions

- rsmi_status_t rsmi_dev_ecc_count_get (uint32_t dv_ind, rsmi_gpu_block_t block, rsmi_error_count_t *ec)

  Retrieve the error counts for a GPU block.
- rsmi_status_t rsmi_dev_ecc_enabled_get (uint32_t dv_ind, uint64_t *enabled_blocks)

  Retrieve the enabled ECC bit-mask.
- rsmi_status_t rsmi_dev_ecc_status_get (uint32_t dv_ind, rsmi_gpu_block_t block, rsmi_ras_err_state_t *state)

  Retrieve the ECC status for a GPU block.
- rsmi_status_t rsmi_status_string (rsmi_status_t status, const char **status_string)

  Get a description of a provided RSMI error status.

5.13.1 Detailed Description

These functions provide error information about RSMI calls as well as device errors.

5.13.2 Function Documentation

5.13.2.1 rsmi_dev_ecc_count_get()

```
rsmi_status_t rsmi_dev_ecc_count_get (
            uint32_t dv_ind,
            rsmi_gpu_block_t block,
            rsmi_error_count_t * ec )
```

Retrieve the error counts for a GPU block.

Given a device index `dv_ind`, an rsmi_gpu_block_t `block` and a pointer to an rsmi_error_count_t `ec`, this function will write the error count values for the GPU block indicated by `block` to memory pointed to by `ec`.

Parameters

in	*dv_ind*	a device index
in	*block*	The block for which error counts should be retrieved
in,out	*ec*	A pointer to an rsmi_error_count_t to which the error counts should be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.13.2.2 rsmi_dev_ecc_enabled_get()

```
rsmi_status_t rsmi_dev_ecc_enabled_get (
            uint32_t dv_ind,
            uint64_t * enabled_blocks )
```

Retrieve the enabled ECC bit-mask.

Given a device index `dv_ind`, and a pointer to a uint64_t `enabled_mask`, this function will write bits to memory pointed to by `enabled_blocks`. Upon a successful call, `enabled_blocks` can then be AND'd with elements of the rsmi_gpu_block_t ennumeration to determine if the corresponding block has ECC enabled. Note that whether a block has ECC enabled or not in the device is independent of whether there is kernel support for error counting for that block. Although a block may be enabled, but there may not be kernel support for reading error counters for that block.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*enabled_blocks*	A pointer to a uint64_t to which the enabled blocks bits will be written. If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.13.2.3 rsmi_dev_ecc_status_get()

```
rsmi_status_t rsmi_dev_ecc_status_get (
            uint32_t dv_ind,
            rsmi_gpu_block_t block,
            rsmi_ras_err_state_t * state )
```

Retrieve the ECC status for a GPU block.

Given a device index `dv_ind`, an rsmi_gpu_block_t `block` and a pointer to an rsmi_ras_err_state_t `state`, this function will write the current state for the GPU block indicated by `block` to memory pointed to by `state`.

Parameters

in	*dv_ind*	a device index
in	*block*	The block for which error counts should be retrieved
in,out	*state*	A pointer to an rsmi_ras_err_state_t to which the ECC state should be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.13.2.4 rsmi_status_string()

```
rsmi_status_t rsmi_status_string (
            rsmi_status_t status,
            const char ** status_string )
```

Get a description of a provided RSMI error status.

Set the provided pointer to a const char *, `status_string`, to a string containing a description of the provided error code `status`.

Parameters

in	*status*	The error status for which a description is desired
in,out	*status_string*	A pointer to a const char * which will be made to point to a description of the provided error code

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call

5.14 Performance Counter Functions

These functions are used to configure, query and control performance counting.

Functions

- rsmi_status_t rsmi_dev_counter_group_supported (uint32_t dv_ind, rsmi_event_group_t group)

  Tell if an event group is supported by a given device.
- rsmi_status_t rsmi_dev_counter_create (uint32_t dv_ind, rsmi_event_type_t type, rsmi_event_handle_t *evnt_handle)

  Create a performance counter object.
- rsmi_status_t rsmi_dev_counter_destroy (rsmi_event_handle_t evnt_handle)

  Deallocate a performance counter object.
- rsmi_status_t rsmi_counter_control (rsmi_event_handle_t evt_handle, rsmi_counter_command_t cmd, void *cmd_args)

  Issue performance counter control commands.
- rsmi_status_t rsmi_counter_read (rsmi_event_handle_t evt_handle, rsmi_counter_value_t *value)

  Read the current value of a performance counter.
- rsmi_status_t rsmi_counter_available_counters_get (uint32_t dv_ind, rsmi_event_group_t grp, uint32_t *available)

  Get the number of currently available counters.

5.14.1 Detailed Description

These functions are used to configure, query and control performance counting.

These functions use the same mechanisms as the "perf" command line utility. They share the same underlying resources and have some similarities in how they are used. The events supported by this API should have corresponding perf events that can be seen with "perf stat ...". The events supported by perf can be seen with "perf list"

The types of events available and the ability to count those events are dependent on which device is being targeted and if counters are still available for that device, respectively. rsmi_dev_counter_group_supported() can be used to see which event types (rsmi_event_group_t) are supported for a given device. Assuming a device supports a given event type, we can then check to see if there are counters available to count a specific event with rsmi_counter_available_counters_get(). Counters may be occupied by other perf based programs.

Once it is determined that events are supported and counters are available, an event counter can be created/destroyed and controlled.

rsmi_dev_counter_create() allocates internal data structures that will be used to used to control the event counter, and return a handle to this data structure.

Once an event counter handle is obtained, the event counter can be controlled (i.e., started, stopped,...) with rsmi_counter_control() by passing rsmi_counter_command_t commands. RSMI_CNTR_CMD_START starts an event counter and RSMI_CNTR_CMD_STOP stops a counter. rsmi_counter_read() reads an event counter.

Once the counter is no longer needed, the resources it uses should be freed by calling rsmi_dev_counter_destroy().

5.14.2 Important Notes about Counter Values

- A running "absolute" counter is kept internally. For the discussion that follows, we will call the internal counter value at time *t* val_t
- Issuing RSMI_CNTR_CMD_START or calling rsmi_counter_read(), causes RSMI (in kernel) to internally record the current absolute counter value
- rsmi_counter_read() returns the number of events that have occurred since the previously recorded value (ie, a relative value, val_t - val_{t-1}) from the issuing of RSMI_CNTR_CMD_START or calling rsmi_counter_read()

Example of event counting sequence:

```
rsmi_counter_value_t value;
// Determine if RSMI_EVNT_GRP_XGMI is supported for device dv_ind
ret = rsmi_dev_counter_group_supported(dv_ind, RSMI_EVNT_GRP_XGMI);
// See if there are counters available for device dv_ind for event
// RSMI_EVNT_GRP_XGMI
ret = rsmi_counter_available_counters_get(dv_ind,
                          RSMI_EVNT_GRP_XGMI, &counters_available);
// Assuming RSMI_EVNT_GRP_XGMI is supported and there is at least 1
// counter available for RSMI_EVNT_GRP_XGMI on device dv_ind, create
// an event object and get the handle (rsmi_event_handle_t).
ret = rsmi_dev_counter_create(dv_ind, RSMI_EVNT_GRP_XGMI, &evnt_handle);
// A program that generates the events of interest can be started
// immediately before or after starting the counters.
// Start counting:
ret = rsmi_counter_control(evnt_handle, RSMI_CNTR_CMD_START, NULL);
// Wait...
// Get the number of events since RSMI_CNTR_CMD_START was issued:
ret = rsmi_counter_read(rsmi_event_handle_t evt_handle, &value)
// Wait...
// Get the number of events since rsmi_counter_read() was last called:
ret = rsmi_counter_read(rsmi_event_handle_t evt_handle, &value)
// Stop counting.
ret = rsmi_counter_control(evnt_handle, RSMI_CNTR_CMD_STOP, NULL);
// Release all resources (e.g., counter and memory resources) associated
with evnt_handle.
ret = rsmi_dev_counter_destroy(evnt_handle);
```

5.14.3 Function Documentation

5.14.3.1 rsmi_dev_counter_group_supported()

```
rsmi_status_t rsmi_dev_counter_group_supported (
              uint32_t dv_ind,
              rsmi_event_group_t group )
```

Tell if an event group is supported by a given device.

Given a device index `dv_ind` and an event group specifier `group`, tell if `group` type events are supported by the device associated with `dv_ind`

Parameters

in	*dv_ind*	device index of device being queried
in	*group*	rsmi_event_group_t identifier of group for which support is being queried

Return values

RSMI_STATUS_SUCCESS	if the device associatee with `dv_ind` support counting events of the type indicated by `group`.
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments group

5.14.3.2 rsmi_dev_counter_create()

```
rsmi_status_t rsmi_dev_counter_create (
              uint32_t dv_ind,
```

```
            rsmi_event_type_t type,
            rsmi_event_handle_t * evnt_handle )
```

Create a performance counter object.

Create a performance counter object of type `type` for the device with a device index of `dv_ind`, and write a handle to the object to the memory location pointed to by `evnt_handle`. `evnt_handle` can be used with other performance event operations. The handle should be deallocated with rsmi_dev_counter_destroy() when no longer needed.

Parameters

in	*dv_ind*	a device index
in	*type*	the rsmi_event_type_t of performance event to create
in,out	*evnt_handle*	A pointer to a rsmi_event_handle_t which will be associated with a newly allocated counter If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_OUT_OF_RESOURCES	unable to allocate memory for counter
RSMI_STATUS_PERMISSION	function requires root access

5.14.3.3 rsmi_dev_counter_destroy()

```
rsmi_status_t rsmi_dev_counter_destroy (
            rsmi_event_handle_t evnt_handle )
```

Deallocate a performance counter object.

Deallocate the performance counter object with the provided rsmi_event_handle_t `evnt_handle`

Parameters

in	*evnt_handle*	handle to event object to be deallocated

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_PERMISSION	function requires root access

5.14.3.4 rsmi_counter_control()

```
rsmi_status_t rsmi_counter_control (
            rsmi_event_handle_t evt_handle,
            rsmi_counter_command_t cmd,
            void * cmd_args )
```

Issue performance counter control commands.

Issue a command `cmd` on the event counter associated with the provided handle `evt_handle`.

Parameters

in	*evt_handle*	an event handle
in	*cmd*	The event counter command to be issued
in,out	*cmd_args*	Currently not used. Should be set to NULL.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_PERMISSION	function requires root access

5.14.3.5 rsmi_counter_read()

```
rsmi_status_t rsmi_counter_read (
            rsmi_event_handle_t evt_handle,
            rsmi_counter_value_t * value )
```

Read the current value of a performance counter.

Read the current counter value of the counter associated with the provided handle `evt_handle` and write the value to the location pointed to by `value`.

Parameters

in	*evt_handle*	an event handle
in,out	*value*	pointer to memory of size of rsmi_counter_value_t to which the counter value will be written

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_PERMISSION	function requires root access

5.14.3.6 rsmi_counter_available_counters_get()

```
rsmi_status_t rsmi_counter_available_counters_get (
            uint32_t dv_ind,
            rsmi_event_group_t grp,
            uint32_t * available )
```

Get the number of currently available counters.

Given a device index `dv_ind`, a performance event group `grp`, and a pointer to a uint32_t `available`, this function will write the number of `grp` type counters that are available on the device with index `dv_ind` to the memory that `available` points to.

Parameters

`in`	*dv_ind*	a device index
`in`	*grp*	an event device group
`in,out`	*available*	A pointer to a uint32_t to which the number of available counters will be written

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.15 System Information Functions

These functions are used to configure, query and control performance counting.

Functions

- rsmi_status_t rsmi_compute_process_info_get (rsmi_process_info_t *procs, uint32_t *num_items)

  Get process information about processes currently using GPU.
- rsmi_status_t rsmi_compute_process_info_by_pid_get (uint32_t pid, rsmi_process_info_t *proc)

  Get process information about a specific process.
- rsmi_status_t rsmi_compute_process_gpus_get (uint32_t pid, uint32_t *dv_indices, uint32_t *num_devices)

  Get the device indices currently being used by a process.

5.15.1 Detailed Description

These functions are used to configure, query and control performance counting.

5.15.2 Function Documentation

5.15.2.1 rsmi_compute_process_info_get()

```
rsmi_status_t rsmi_compute_process_info_get (
            rsmi_process_info_t * procs,
            uint32_t * num_items )
```

Get process information about processes currently using GPU.

Given a non-NULL pointer to an array `procs` of rsmi_process_info_t's, of length `*num_items`, this function will write up to `*num_items` instances of rsmi_process_info_t to the memory pointed to by `procs`. These instances contain information about each process utilizing a GPU. If `procs` is not NULL, `num_items` will be updated with the number of processes actually written. If `procs` is NULL, `num_items` will be updated with the number of processes for which there is current process information. Calling this function with `procs` being NULL is a way to determine how much memory should be allocated for when `procs` is not NULL.

Parameters

`in,out`	*procs*	a pointer to memory provided by the caller to which process information will be written. This may be NULL in which case only `num_items` will be updated with the number of processes found.
`in,out`	*num_items*	A pointer to a uint32_t, which on input, should contain the amount of memory in rsmi_process_info_t's which have been provided by the `procs` argument. On output, if `procs` is non-NULL, this will be updated with the number rsmi_process_info_t structs actually written. If `procs` is NULL, this argument will be updated with the number processes for which there is information.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if there were more processes for which information was available, but not enough space was provided as indicated by `procs` and `num_items`, on input.

5.15.2.2 rsmi_compute_process_info_by_pid_get()

```
rsmi_status_t rsmi_compute_process_info_by_pid_get (
            uint32_t pid,
            rsmi_process_info_t * proc )
```

Get process information about a specific process.

Given a pointer to an rsmi_process_info_t `proc` and a process id `pid`, this function will write the process information for `pid`, if available, to the memory pointed to by `proc`.

Parameters

`in`	*pid*	The process ID for which process information is being requested
`in,out`	*proc*	a pointer to a rsmi_process_info_t to which process information for `pid` will be written if it is found.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_NOT_FOUND	is returned if there was no process information found for the provided `pid`

5.15.2.3 rsmi_compute_process_gpus_get()

```
rsmi_status_t rsmi_compute_process_gpus_get (
            uint32_t pid,
            uint32_t * dv_indices,
            uint32_t * num_devices )
```

Get the device indices currently being used by a process.

Given a process id `pid`, a non-NULL pointer to an array of uint32_t's `dv_indices` of length `*num_devices`, this function will write up to `num_devices` device indices to the memory pointed to by `dv_indices`. If `dv_indices` is not NULL, `num_devices` will be updated with the number of gpu's currently being used by process `pid`. If `dv_indices` is NULL, `dv_indices` will be updated with the number of gpus currently being used by `pid`. Calling this function with `dv_indices` being NULL is a way to determine how much memory is required for when `dv_indices` is not NULL.

Parameters

in	*pid*	The process id of the process for which the number of gpus currently being used is requested
in,out	*dv_indices*	a pointer to memory provided by the caller to which indices of devices currently being used by the process will be written. This may be NULL in which case only num_devices will be updated with the number of devices being used.
in,out	*num_devices*	A pointer to a uint32_t, which on input, should contain the amount of memory in uint32_t's which have been provided by the dv_indices argument. On output, if dv_indices is non-NULL, this will be updated with the number uint32_t's actually written. If dv_indices is NULL, this argument will be updated with the number devices being used.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid
RSMI_STATUS_INSUFFICIENT_SIZE	is returned if there were more gpu indices that could have been written, but not enough space was provided as indicated by dv_indices and num_devices, on input.

5.16 XGMI Functions

These functions are used to configure, query and control XGMI.

Functions

- rsmi_status_t rsmi_dev_xgmi_error_status (uint32_t dv_ind, rsmi_xgmi_status_t *status)

  Retrieve the XGMI error status for a device.
- rsmi_status_t rsmi_dev_xgmi_error_reset (uint32_t dv_ind)

  Reset the XGMI error status for a device.
- rsmi_status_t rsmi_dev_xgmi_hive_id_get (uint32_t dv_ind, uint64_t *hive_id)

  Retrieve the XGMI hive id for a device.

5.16.1 Detailed Description

These functions are used to configure, query and control XGMI.

5.16.2 Function Documentation

5.16.2.1 rsmi_dev_xgmi_error_status()

```
rsmi_status_t rsmi_dev_xgmi_error_status (
            uint32_t dv_ind,
            rsmi_xgmi_status_t * status )
```

Retrieve the XGMI error status for a device.

Given a device index `dv_ind`, and a pointer to an rsmi_xgmi_status_t `status`, this function will write the current XGMI error state rsmi_xgmi_status_t for the device `dv_ind` to the memory pointed to by `status`.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*status*	A pointer to an rsmi_xgmi_status_t to which the XGMI error state should be written If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.16.2.2 rsmi_dev_xgmi_error_reset()

```
rsmi_status_t rsmi_dev_xgmi_error_reset (
            uint32_t dv_ind )
```

Reset the XGMI error status for a device.

Given a device index `dv_ind`, this function will reset the current XGMI error state rsmi_xgmi_status_t for the device `dv_ind` to rsmi_xgmi_status_t::RSMI_XGMI_STATUS_NO_ERRORS

Parameters

`in`	*dv_ind*	a device index

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

5.16.2.3 rsmi_dev_xgmi_hive_id_get()

```
rsmi_status_t rsmi_dev_xgmi_hive_id_get (
            uint32_t dv_ind,
            uint64_t * hive_id )
```

Retrieve the XGMI hive id for a device.

Given a device index `dv_ind`, and a pointer to an uint64_t `hive_id`, this function will write the current XGMI hive id for the device `dv_ind` to the memory pointed to by `hive_id`.

Parameters

`in`	*dv_ind*	a device index
`in,out`	*hive_id*	A pointer to an uint64_t to which the XGMI hive id should be written

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.17 Hardware Topology Functions

These functions are used to query Hardware topology.

Functions

- rsmi_status_t rsmi_topo_get_numa_node_number (uint32_t dv_ind, uint32_t *numa_node)

  Retrieve the NUMA CPU node number for a device.
- rsmi_status_t rsmi_topo_get_link_weight (uint32_t dv_ind_src, uint32_t dv_ind_dst, uint64_t *weight)

  Retrieve the weight for a connection between 2 GPUs.
- rsmi_status_t rsmi_topo_get_link_type (uint32_t dv_ind_src, uint32_t dv_ind_dst, uint64_t *hops, RSMI_IO_LINK_TYPE *type)

  Retrieve the hops and the connection type between 2 GPUs.

5.17.1 Detailed Description

These functions are used to query Hardware topology.

5.17.2 Function Documentation

5.17.2.1 rsmi_topo_get_numa_node_number()

```
rsmi_status_t rsmi_topo_get_numa_node_number (
            uint32_t dv_ind,
            uint32_t * numa_node )
```

Retrieve the NUMA CPU node number for a device.

Given a device index `dv_ind`, and a pointer to an uint32_t `numa_node`, this function will write the node number of NUMA CPU for the device `dv_ind` to the memory pointed to by `numa_node`.

Parameters

in	*dv_ind*	a device index
in,out	*numa_node*	A pointer to an uint32_t to which the numa node number should be written.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.17.2.2 rsmi_topo_get_link_weight()

```
rsmi_status_t rsmi_topo_get_link_weight (
            uint32_t dv_ind_src,
            uint32_t dv_ind_dst,
            uint64_t * weight )
```

Retrieve the weight for a connection between 2 GPUs.

Given a source device index `dv_ind_src` and a destination device index `dv_ind_dst`, and a pointer to an uint64_t `weight`, this function will write the weight for the connection between the device `dv_ind_src` and `dv_ind_dst` to the memory pointed to by `weight`.

Parameters

in	*dv_ind_src*	the source device index
in	*dv_ind_dst*	the destination device index
in,out	*weight*	A pointer to an uint64_t to which the weight for the connection should be written.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.17.2.3 rsmi_topo_get_link_type()

```
rsmi_status_t rsmi_topo_get_link_type (
            uint32_t dv_ind_src,
            uint32_t dv_ind_dst,
            uint64_t * hops,
            RSMI_IO_LINK_TYPE * type )
```

Retrieve the hops and the connection type between 2 GPUs.

Given a source device index `dv_ind_src` and a destination device index `dv_ind_dst`, and a pointer to an uint64_t `hops` and a pointer to an RSMI_IO_LINK_TYPE `type`, this function will write the number of hops and the connection type between the device `dv_ind_src` and `dv_ind_dst` to the memory pointed to by `hops` and `type`.

Parameters

in	*dv_ind_src*	the source device index
in	*dv_ind_dst*	the destination device index
in,out	*hops*	A pointer to an uint64_t to which the hops for the connection should be written.
in,out	*type*	A pointer to an RSMI_IO_LINK_TYPE to which the type for the connection should be written.

Return values

RSMI_STATUS_SUCCESS	call was successful

Return values

RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

5.18 Supported Functions

API function support varies by both GPU type and the version of the installed ROCm stack. The functions described in this section can be used to determine, up front, which functions are supported for a given device on a system. If such "up front" knowledge of support for a function is not needed, alternatively, one can call a device related function and check the return code.

Functions

- rsmi_status_t rsmi_dev_supported_func_iterator_open (uint32_t dv_ind, rsmi_func_id_iter_handle_t *handle)

  Get a function name iterator of supported RSMI functions for a device.
- rsmi_status_t rsmi_dev_supported_variant_iterator_open (rsmi_func_id_iter_handle_t obj_h, rsmi_func_id_iter_handle_t *var_iter)

  Get a variant iterator for a given handle.
- rsmi_status_t rsmi_func_iter_next (rsmi_func_id_iter_handle_t handle)

  Advance a function identifer iterator.
- rsmi_status_t rsmi_dev_supported_func_iterator_close (rsmi_func_id_iter_handle_t *handle)

  Close a variant iterator handle.
- rsmi_status_t rsmi_func_iter_value_get (rsmi_func_id_iter_handle_t handle, rsmi_func_id_value_t *value)

  Get the value associated with a function/variant iterator.

5.18.1 Detailed Description

API function support varies by both GPU type and the version of the installed ROCm stack. The functions described in this section can be used to determine, up front, which functions are supported for a given device on a system. If such "up front" knowledge of support for a function is not needed, alternatively, one can call a device related function and check the return code.

Some functions have several variations ("variants") where some variants are supported and others are not. For example, on a given device, rsmi_dev_temp_metric_get may support some types of temperature metrics (e.g., RSMI_TEMP_CRITICAL_HYST), but not others (e.g., RSMI_TEMP_EMERGENCY).

In addition to a top level of variant support for a function, a function may have varying support for monitors/sensors. These are considered "sub-variants" in functions described in this section. Continuing the rsmi_dev_temp_metric_get example, if variant RSMI_TEMP_CRITICAL_HYST is supported, perhaps only the sub-variant sensors RSMI_TEMP_TYPE_EDGE and RSMI_TEMP_TYPE_EDGE are supported, but not RSMI_TEMP_TYPE_MEMORY.

In cases where a function takes in a sensor id parameter but does not have any "top level" variants, the functions in this section will indicate a default "variant", RSMI_DEFAULT_VARIANT, for the top level variant, and the various monitor support will be sub-variants of this.

The functions in this section use the "iterator" concept to list which functions are supported; to list which variants of the supported functions are supported; and finally which monitors/sensors are supported for a variant.

Here is example code that prints out all supported functions, their supported variants and sub-variants. Please see the related descriptions functions and RSMI types.

```
rsmi_func_id_iter_handle_t iter_handle, var_iter, sub_var_iter;
rsmi_func_id_value_t value;
rsmi_status_t err;
for (uint32_t i = 0; i < <number of devices>; ++i) {
 std::cout « "Supported RSMI Functions:" « std::endl;
 std::cout « "\tVariants (Monitors)" « std::endl;
 err = rsmi_dev_supported_func_iterator_open(i, &iter_handle);
 while (1) {
   err = rsmi_func_iter_value_get(iter_handle, &value);
   std::cout « "Function Name: " « value.name « std::endl;
   err = rsmi_dev_supported_variant_iterator_open(iter_handle, &var_iter);
   if (err != RSMI_STATUS_NO_DATA) {
     std::cout « "\tVariants/Monitors: ";
     while (1) {
       err = rsmi_func_iter_value_get(var_iter, &value);
       if (value.id == RSMI_DEFAULT_VARIANT) {
         std::cout « "Default Variant ";
       } else {
         std::cout « value.id;
       }
       std::cout « " (";
       err =
         rsmi_dev_supported_variant_iterator_open(var_iter, &sub_var_iter);
       if (err != RSMI_STATUS_NO_DATA) {
         while (1) {
           err = rsmi_func_iter_value_get(sub_var_iter, &value);
           std::cout « value.id « ", ";
           err = rsmi_func_iter_next(sub_var_iter);
           if (err == RSMI_STATUS_NO_DATA) {
             break;
           }
         }
         err = rsmi_dev_supported_func_iterator_close(&sub_var_iter);
       }
       std::cout « "), ";
       err = rsmi_func_iter_next(var_iter);
       if (err == RSMI_STATUS_NO_DATA) {
         break;
       }
     }
     std::cout « std::endl;
     err = rsmi_dev_supported_func_iterator_close(&var_iter);
   }
   err = rsmi_func_iter_next(iter_handle);
   if (err == RSMI_STATUS_NO_DATA) {
     break;
   }
 }
 err = rsmi_dev_supported_func_iterator_close(&iter_handle);
}
```

5.18.2 Function Documentation

5.18.2.1 rsmi_dev_supported_func_iterator_open()

```
rsmi_status_t rsmi_dev_supported_func_iterator_open (
            uint32_t dv_ind,
            rsmi_func_id_iter_handle_t * handle )
```

Get a function name iterator of supported RSMI functions for a device.

Given a device index `dv_ind`, this function will write a function iterator handle to the caller-provided memory pointed to by `handle`. This handle can be used to iterate through all the supported functions.

Note that although this function takes in `dv_ind` as an argument, rsmi_dev_supported_func_iterator_open itself will not be among the functions listed as supported. This is because rsmi_dev_supported_func_iterator_open does not depend on hardware or driver support and should always be supported.

Parameters

in	*dv_ind*	a device index of device for which support information is requested
in,out	*handle*	A pointer to caller-provided memory to which the function iterator will be written.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

5.18.2.2 rsmi_dev_supported_variant_iterator_open()

```
rsmi_status_t rsmi_dev_supported_variant_iterator_open (
            rsmi_func_id_iter_handle_t obj_h,
            rsmi_func_id_iter_handle_t * var_iter )
```

Get a variant iterator for a given handle.

Given a rsmi_func_id_iter_handle_t `obj_h`, this function will write a function iterator handle to the caller-provided memory pointed to by `var_iter`. This handle can be used to iterate through all the supported variants of the provided handle. `obj_h` may be a handle to a function object, as provided by a call to rsmi_dev_supported_func_iterator_open, or it may be a variant itself (from a call to rsmi_dev_supported_variant_iterator_open), it which case `var_iter` will be an iterator of the sub-variants of `obj_h` (e.g., monitors).

This call allocates a small amount of memory to `var_iter`. To free this memory rsmi_dev_supported_func_iterator_close should be called on the returned iterator handle `var_iter` when it is no longer needed.

Parameters

in	*obj_h*	an iterator handle for which the variants are being requested
in,out	*var_iter*	A pointer to caller-provided memory to which the sub-variant iterator will be written.

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

5.18.2.3 rsmi_func_iter_next()

```
rsmi_status_t rsmi_func_iter_next (
            rsmi_func_id_iter_handle_t handle )
```

Advance a function identifer iterator.

Given a function id iterator handle (rsmi_func_id_iter_handle_t) `handle`, this function will increment the iterator to point to the next identifier. After a successful call to this function, obtaining the value of the iterator `handle` will provide the value of the next item in the list of functions/variants.

If there are no more items in the list, RSMI_STATUS_NO_DATA is returned.

Parameters

in	*handle*	A pointer to an iterator handle to be incremented

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.
RSMI_STATUS_NO_DATA	is returned when list of identifiers has been exhausted

5.18.2.4 rsmi_dev_supported_func_iterator_close()

```
rsmi_status_t rsmi_dev_supported_func_iterator_close (
            rsmi_func_id_iter_handle_t * handle )
```

Close a variant iterator handle.

Given a pointer to an rsmi_func_id_iter_handle_t `handle`, this function will free the resources being used by the handle

Parameters

in	*handle*	A pointer to an iterator handle to be closed

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

5.18.2.5 rsmi_func_iter_value_get()

```
rsmi_status_t rsmi_func_iter_value_get (
            rsmi_func_id_iter_handle_t handle,
            rsmi_func_id_value_t * value )
```

Get the value associated with a function/variant iterator.

Given an rsmi_func_id_iter_handle_t `handle`, this function will write the identifier of the function/variant to the user provided memory pointed to by `value`.

`value` may point to a function name, a variant id, or a monitor/sensor index, depending on what kind of iterator `handle` is

Parameters

in	*handle*	An iterator for which the value is being requested
in, out	*value*	A pointer to an rsmi_func_id_value_t provided by the caller to which this function will write the value assocaited with `handle`

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

Chapter 6

Data Structure Documentation

6.1 id Union Reference

This union holds the value of an rsmi_func_id_iter_handle_t. The value may be a function name, or an ennumerated variant value of types such as rsmi_memory_type_t, rsmi_temperature_metric_t, etc.

`#include <rocm_smi.h>`

Data Fields

- uint64_t id

  uint64_t representation of value
- const char * name

  name string (applicable to functions only)
-

  union {
  rsmi_memory_type_t memory_type
  < *Used for rsmi_memory_type_t variants*
  rsmi_temperature_metric_t temp_metric
  Used for rsmi_event_type_t variants.
  rsmi_event_type_t evnt_type
  Used for rsmi_event_group_t variants.
  rsmi_event_group_t evnt_group
  Used for rsmi_clk_type_t variants.
  rsmi_clk_type_t clk_type
  Used for rsmi_fw_block_t variants.
  rsmi_fw_block_t fw_block
  Used for rsmi_gpu_block_t variants.
  rsmi_gpu_block_t **gpu_block_type**
  };

6.1.1 Detailed Description

This union holds the value of an rsmi_func_id_iter_handle_t. The value may be a function name, or an ennumerated variant value of types such as rsmi_memory_type_t, rsmi_temperature_metric_t, etc.

6.1.2 Field Documentation

6.1.2.1 memory_type

```
rsmi_memory_type_t id::memory_type
```

< Used for rsmi_memory_type_t variants

Used for rsmi_temperature_metric_t variants

The documentation for this union was generated from the following file:

- rocm_smi.h

6.2 rsmi_counter_value_t Struct Reference

```
#include <rocm_smi.h>
```

Data Fields

- uint64_t value
  Counter value.
- uint64_t time_enabled
- uint64_t time_running

6.2.1 Detailed Description

Counter value

6.2.2 Field Documentation

6.2.2.1 time_enabled

```
uint64_t rsmi_counter_value_t::time_enabled
```

Time that the counter was enabled (in nanoseconds)

6.2.2.2 time_running

```
uint64_t rsmi_counter_value_t::time_running
```

Time that the counter was running (in nanoseconds)

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.3 rsmi_error_count_t Struct Reference

This structure holds error counts.

```
#include <rocm_smi.h>
```

Data Fields

- uint64_t correctable_err
  Accumulated correctable errors.
- uint64_t uncorrectable_err
  Accumulated uncorrectable errors.

6.3.1 Detailed Description

This structure holds error counts.

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.4 rsmi_evt_notification_data_t Struct Reference

```
#include <rocm_smi.h>
```

Data Fields

- uint32_t dv_ind
  Index of device that corresponds to the event.
- rsmi_evt_notification_type_t event
  Event type.
- char message [MAX_EVENT_NOTIFICATION_MSG_SIZE]
  Event message.

6.4.1 Detailed Description

Event notification data returned from event notification API

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.5 rsmi_freq_volt_region_t Struct Reference

This structure holds 2 rsmi_range_t's, one for frequency and one for voltage. These 2 ranges indicate the range of possible values for the corresponding rsmi_od_vddc_point_t.

`#include <rocm_smi.h>`

Data Fields

- rsmi_range_t freq_range

  The frequency range for this VDDC Curve point.
- rsmi_range_t volt_range

  The voltage range for this VDDC Curve point.

6.5.1 Detailed Description

This structure holds 2 rsmi_range_t's, one for frequency and one for voltage. These 2 ranges indicate the range of possible values for the corresponding rsmi_od_vddc_point_t.

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.6 rsmi_frequencies_t Struct Reference

This structure holds information about clock frequencies.

`#include <rocm_smi.h>`

Data Fields

- uint32_t num_supported
- uint32_t current
- uint64_t frequency [RSMI_MAX_NUM_FREQUENCIES]

6.6.1 Detailed Description

This structure holds information about clock frequencies.

6.6.2 Field Documentation

6.6.2.1 num_supported

```
uint32_t rsmi_frequencies_t::num_supported
```

The number of supported frequencies

6.6.2.2 current

```
uint32_t rsmi_frequencies_t::current
```

The current frequency index

6.6.2.3 frequency

```
uint64_t rsmi_frequencies_t::frequency[RSMI_MAX_NUM_FREQUENCIES]
```

List of frequencies. Only the first num_supported frequencies are valid.

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.7 rsmi_od_vddc_point_t Struct Reference

This structure represents a point on the frequency-voltage plane.

```
#include <rocm_smi.h>
```

Data Fields

- uint64_t frequency

  Frequency coordinate (in Hz)
- uint64_t voltage

  Voltage coordinate (in mV)

6.7.1 Detailed Description

This structure represents a point on the frequency-voltage plane.

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.8 rsmi_od_volt_curve_t Struct Reference

`#include <rocm_smi.h>`

Data Fields

- rsmi_od_vddc_point_t vc_points [RSMI_NUM_VOLTAGE_CURVE_POINTS]

6.8.1 Detailed Description

RSMI_NUM_VOLTAGE_CURVE_POINTS number of rsmi_od_vddc_point_t's

6.8.2 Field Documentation

6.8.2.1 vc_points

`rsmi_od_vddc_point_t rsmi_od_volt_curve_t::vc_points[RSMI_NUM_VOLTAGE_CURVE_POINTS]`

Array of RSMI_NUM_VOLTAGE_CURVE_POINTS rsmi_od_vddc_point_t's that make up the voltage frequency curve points.

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.9 rsmi_od_volt_freq_data_t Struct Reference

This structure holds the frequency-voltage values for a device.

`#include <rocm_smi.h>`

Data Fields

- rsmi_range_t curr_sclk_range

  The current SCLK frequency range.
- rsmi_range_t curr_mclk_range
- rsmi_range_t sclk_freq_limits

  The range possible of SCLK values.
- rsmi_range_t mclk_freq_limits

  The range possible of MCLK values.
- rsmi_od_volt_curve_t curve

  The current voltage curve.
- uint32_t num_regions

  The number of voltage curve regions.

6.9.1 Detailed Description

This structure holds the frequency-voltage values for a device.

6.9.2 Field Documentation

6.9.2.1 curr_mclk_range

```
rsmi_range_t rsmi_od_volt_freq_data_t::curr_mclk_range
```

The current MCLK frequency range; (upper bound only)

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.10 rsmi_pcie_bandwidth_t Struct Reference

This structure holds information about the possible PCIe bandwidths. Specifically, the possible transfer rates and their associated numbers of lanes are stored here.

```
#include <rocm_smi.h>
```

Data Fields

- rsmi_frequencies_t transfer_rate
- uint32_t lanes [RSMI_MAX_NUM_FREQUENCIES]

6.10.1 Detailed Description

This structure holds information about the possible PCIe bandwidths. Specifically, the possible transfer rates and their associated numbers of lanes are stored here.

6.10.2 Field Documentation

6.10.2.1 transfer_rate

```
rsmi_frequencies_t rsmi_pcie_bandwidth_t::transfer_rate
```

Transfer rates (T/s) that are possible

6.10.2.2 lanes

```
uint32_t rsmi_pcie_bandwidth_t::lanes[RSMI_MAX_NUM_FREQUENCIES]
```

List of lanes for corresponding transfer rate. Only the first num_supported bandwidths are valid.

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.11 rsmi_power_profile_status_t Struct Reference

This structure contains information about which power profiles are supported by the system for a given device, and which power profile is currently active.

```
#include <rocm_smi.h>
```

Data Fields

- rsmi_bit_field_t available_profiles
- rsmi_power_profile_preset_masks_t current
- uint32_t num_profiles

6.11.1 Detailed Description

This structure contains information about which power profiles are supported by the system for a given device, and which power profile is currently active.

6.11.2 Field Documentation

6.11.2.1 available_profiles

```
rsmi_bit_field_t rsmi_power_profile_status_t::available_profiles
```

Which profiles are supported by this system

6.11.2.2 current

```
rsmi_power_profile_preset_masks_t rsmi_power_profile_status_t::current
```

Which power profile is currently active

6.11.2.3 num_profiles

```
uint32_t rsmi_power_profile_status_t::num_profiles
```

How many power profiles are available

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.12 rsmi_process_info_t Struct Reference

This structure contains information specific to a process.

```
#include <rocm_smi.h>
```

Data Fields

- uint32_t process_id
  Process ID.
- uint32_t pasid
  PASID.
- uint64_t vram_usage
  VRAM usage.
- uint64_t sdma_usage
  SDMA usage in microseconds.

6.12.1 Detailed Description

This structure contains information specific to a process.

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.13 rsmi_range_t Struct Reference

This structure represents a range (e.g., frequencies or voltages).

`#include <rocm_smi.h>`

Data Fields

- uint64_t lower_bound
  Lower bound of range.
- uint64_t upper_bound
  Upper bound of range.

6.13.1 Detailed Description

This structure represents a range (e.g., frequencies or voltages).

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.14 rsmi_retired_page_record_t Struct Reference

Reserved Memory Page Record.

`#include <rocm_smi.h>`

Data Fields

- uint64_t page_address
  Start address of page.
- uint64_t page_size
  Page size.
- rsmi_memory_page_status_t status
  Page "reserved" status.

6.14.1 Detailed Description

Reserved Memory Page Record.

The documentation for this struct was generated from the following file:

- rocm_smi.h

6.15 rsmi_version_t Struct Reference

This structure holds version information.

```
#include <rocm_smi.h>
```

Data Fields

- uint32_t major
  Major version.
- uint32_t minor
  Minor version.
- uint32_t patch
  Patch, build or stepping version.
- const char * build
  Build string.

6.15.1 Detailed Description

This structure holds version information.

The documentation for this struct was generated from the following file:

- rocm_smi.h

Chapter 7

File Documentation

7.1 rocm_smi.h File Reference

The rocm_smi library api is new, and therefore subject to change either at the ABI or API level. Instead of marking every function prototype as "unstable", we are instead saying the API is unstable (i.e., changes are possible) while the major version remains 0. This means that if the API/ABI changes, we will not increment the major version to 1. Once the ABI stabilizes, we will increment the major version to 1, and thereafter increment it on all ABI breaks.

```
#include <stdint.h>
#include <stddef.h>
#include "rocm_smi/kfd_ioctl.h"
```

Data Structures

- struct rsmi_counter_value_t
- struct rsmi_evt_notification_data_t
- struct rsmi_retired_page_record_t

  Reserved Memory Page Record.
- struct rsmi_power_profile_status_t

  This structure contains information about which power profiles are supported by the system for a given device, and which power profile is currently active.
- struct rsmi_frequencies_t

  This structure holds information about clock frequencies.
- struct rsmi_pcie_bandwidth_t

  This structure holds information about the possible PCIe bandwidths. Specifically, the possible transfer rates and their associated numbers of lanes are stored here.
- struct rsmi_version_t

  This structure holds version information.
- struct rsmi_range_t

  This structure represents a range (e.g., frequencies or voltages).
- struct rsmi_od_vddc_point_t

  This structure represents a point on the frequency-voltage plane.
- struct rsmi_freq_volt_region_t

  This structure holds 2 rsmi_range_t's, one for frequency and one for voltage. These 2 ranges indicate the range of possible values for the corresponding rsmi_od_vddc_point_t.
- struct rsmi_od_volt_curve_t

- struct rsmi_od_volt_freq_data_t

  This structure holds the frequency-voltage values for a device.
- struct rsmi_error_count_t

  This structure holds error counts.
- struct rsmi_process_info_t

  This structure contains information specific to a process.
- union id

  This union holds the value of an rsmi_func_id_iter_handle_t. The value may be a function name, or an ennumerated variant value of types such as rsmi_memory_type_t, rsmi_temperature_metric_t, etc.

Macros

- #define RSMI_MAX_NUM_FREQUENCIES 32

  Guaranteed maximum possible number of supported frequencies.
- #define RSMI_MAX_FAN_SPEED 255
- #define RSMI_NUM_VOLTAGE_CURVE_POINTS 3

  The number of points that make up a voltage-frequency curve definition.
- #define MAX_EVENT_NOTIFICATION_MSG_SIZE 64

  Maximum number of characters an event notification message will be.
- #define RSMI_MAX_NUM_POWER_PROFILES (sizeof(rsmi_bit_field_t) * 8)

  Number of possible power profiles that a system could support.
- #define RSMI_DEFAULT_VARIANT 0xFFFFFFFFFFFFFFFF

Typedefs

- typedef uintptr_t rsmi_event_handle_t

  Handle to performance event counter.
- typedef uint64_t rsmi_bit_field_t

  Bitfield used in various RSMI calls.
- typedef enum _RSMI_IO_LINK_TYPE RSMI_IO_LINK_TYPE

  Types for IO Link.
- typedef struct rsmi_func_id_iter_handle * rsmi_func_id_iter_handle_t

  Opaque handle to function-support object.
- typedef union id rsmi_func_id_value_t

  This union holds the value of an rsmi_func_id_iter_handle_t. The value may be a function name, or an ennumerated variant value of types such as rsmi_memory_type_t, rsmi_temperature_metric_t, etc.

Enumerations

- enum rsmi_status_t {
  RSMI_STATUS_SUCCESS = 0x0, RSMI_STATUS_INVALID_ARGS, RSMI_STATUS_NOT_SUPPORTED,
  RSMI_STATUS_FILE_ERROR,
  RSMI_STATUS_PERMISSION, RSMI_STATUS_OUT_OF_RESOURCES, RSMI_STATUS_INTERNAL_EXCEPTION,
  RSMI_STATUS_INPUT_OUT_OF_BOUNDS,
  RSMI_STATUS_INIT_ERROR, **RSMI_INITIALIZATION_ERROR** = RSMI_STATUS_INIT_ERROR,
  RSMI_STATUS_NOT_YET_IMPLEMENTED, RSMI_STATUS_NOT_FOUND,
  RSMI_STATUS_INSUFFICIENT_SIZE, RSMI_STATUS_INTERRUPT, RSMI_STATUS_UNEXPECTED_SIZE,
  RSMI_STATUS_NO_DATA,
  RSMI_STATUS_UNEXPECTED_DATA, RSMI_STATUS_BUSY, RSMI_STATUS_REFCOUNT_OVERFLOW,
  RSMI_STATUS_UNKNOWN_ERROR = 0xFFFFFFFF }

Error codes retured by rocm_smi_lib functions.

- enum rsmi_init_flags_t { RSMI_INIT_FLAG_ALL_GPUS = 0x1, RSMI_INIT_FLAG_RESRV_TEST1 = 0x800000000000000 }

  Initialization flags.

- enum rsmi_dev_perf_level_t { RSMI_DEV_PERF_LEVEL_AUTO = 0, **RSMI_DEV_PERF_LEVEL_FIRST** = RSMI_DEV_PERF_LEVEL_AUTO, RSMI_DEV_PERF_LEVEL_LOW, RSMI_DEV_PERF_LEVEL_HIGH, RSMI_DEV_PERF_LEVEL_MANUAL, RSMI_DEV_PERF_LEVEL_STABLE_STD, RSMI_DEV_PERF_LEVEL_STABLE_PEA RSMI_DEV_PERF_LEVEL_STABLE_MIN_MCLK, RSMI_DEV_PERF_LEVEL_STABLE_MIN_SCLK, **RSMI_DEV_PERF_LEVEL_LAST** = RSMI_DEV_PER↩F_LEVEL_STABLE_MIN_SCLK, RSMI_DEV_PERF_LEVEL_UNKNOWN = 0x100 }

  PowerPlay performance levels.

- enum rsmi_sw_component_t { **RSMI_SW_COMP_FIRST** = 0x0, RSMI_SW_COMP_DRIVER = RSMI_SW↩_COMP_FIRST, **RSMI_SW_COMP_LAST** = RSMI_SW_COMP_DRIVER }

  Available clock types.

- enum rsmi_event_group_t { RSMI_EVNT_GRP_XGMI = 0, **RSMI_EVNT_GRP_INVALID** = 0xFFFFFFFF }

  Enum denoting an event group. The value of the enum is the base value for all the event enums in the group.

- enum rsmi_event_type_t { **RSMI_EVNT_FIRST** = RSMI_EVNT_GRP_XGMI, **RSMI_EVNT_XGMI_FIRST** = RSMI_EVNT_GRP_XGMI, RSMI_EVNT_XGMI_0_NOP_TX = RSMI_EVNT_XGMI_FIRST, RSMI_EVNT_XGMI_0_REQUEST_TX, RSMI_EVNT_XGMI_0_RESPONSE_TX, RSMI_EVNT_XGMI_0_BEATS_TX, RSMI_EVNT_XGMI_1_NOP_TX, RSMI_EVNT_XGMI_1_REQUEST_TX, RSMI_EVNT_XGMI_1_RESPONSE_TX, RSMI_EVNT_XGMI_1_BEATS_TX, **RSMI_EVNT_XGMI_LAST** = RSMI_EVNT_XGMI_1_BEATS_TX, **RSMI_EVNT_LAST** = RSMI_EVNT_XGMI_LAST }

  Event type enum. Events belonging to a particular event group rsmi_event_group_t should begin ennumerating at the rsmi_event_group_t value for that group.

- enum rsmi_counter_command_t { RSMI_CNTR_CMD_START = 0, RSMI_CNTR_CMD_STOP }
- enum rsmi_evt_notification_type_t { RSMI_EVT_NOTIF_VMFAULT = KFD_SMI_EVENT_VMFAULT, **RSM↩I_EVT_NOTIF_FIRST** = RSMI_EVT_NOTIF_VMFAULT, **RSMI_EVT_NOTIF_LAST** = RSMI_EVT_NOTIF↩_VMFAULT }
- enum rsmi_clk_type_t { RSMI_CLK_TYPE_SYS = 0x0, **RSMI_CLK_TYPE_FIRST** = RSMI_CLK_TYPE_SYS, RSMI_CLK_TYPE_DF, RSMI_CLK_TYPE_DCEF, RSMI_CLK_TYPE_SOC, RSMI_CLK_TYPE_MEM, **RSMI_CLK_TYPE_LAST** = RSMI_CLK_TYPE_MEM, **RSMI_CLK_INVALID** = 0xFFFFFFFF }
- enum rsmi_temperature_metric_t { RSMI_TEMP_CURRENT = 0x0, **RSMI_TEMP_FIRST** = RSMI_TEMP_CURRENT, RSMI_TEMP_MAX, RSMI_TEMP_MIN, RSMI_TEMP_MAX_HYST, RSMI_TEMP_MIN_HYST, RSMI_TEMP_CRITICAL, RSMI_TEMP_CRITICAL_HYST, RSMI_TEMP_EMERGENCY, RSMI_TEMP_EMERGENCY_HYST, RSMI_TEMP_CRIT_MIN, RSMI_TEMP_CRIT_MIN_HYST RSMI_TEMP_OFFSET, RSMI_TEMP_LOWEST, RSMI_TEMP_HIGHEST, **RSMI_TEMP_LAST** = RSMI_↩TEMP_HIGHEST }

  Temperature Metrics. This enum is used to identify various temperature metrics. Corresponding values will be in millidegress Celcius.

- enum rsmi_temperature_type_t { **RSMI_TEMP_TYPE_FIRST** = 0, RSMI_TEMP_TYPE_EDGE = RSMI_TEMP_TYPE_FIRST, RSMI_TEMP_TYPE_JUNCTION, RSMI_TEMP_TYPE_MEMORY, **RSMI_TEMP_TYPE_LAST** = RSMI_TEMP_TYPE_MEMORY, RSMI_TEMP_TYPE_INVALID = 0xFFFFF↩FFF }

  This ennumeration is used to indicate from which part of the device a temperature reading should be obtained.

- enum rsmi_voltage_metric_t { RSMI_VOLT_CURRENT = 0x0, **RSMI_VOLT_FIRST** = RSMI_VOLT_CURRENT, RSMI_VOLT_MAX, RSMI_VOLT_MIN_CRIT, RSMI_VOLT_MIN, RSMI_VOLT_MAX_CRIT, RSMI_VOLT_AVERAGE, RSMI_VOLT_LOWEST, RSMI_VOLT_HIGHEST, **RSMI_VOLT_LAST** = RSMI_VOLT_HIGHEST }

  Voltage Metrics. This enum is used to identify various Volatge metrics. Corresponding values will be in millivolt.

- enum rsmi_voltage_type_t { **RSMI_VOLT_TYPE_FIRST** = 0, RSMI_VOLT_TYPE_VDDGFX = RSMI_VOLT_TYPE_FIRST, **RSMI_VOLT_TYPE_LAST** = RSMI_VOLT_TYPE_VDDGFX, RSMI_VOLT_TYPE_INVALID = 0xFFFFFFFF }

  This ennumeration is used to indicate which type of voltage reading should be obtained.

- enum rsmi_power_profile_preset_masks_t { RSMI_PWR_PROF_PRST_CUSTOM_MASK = 0x1, RSMI_PWR_PROF_PRST_VIDEO_MASK = 0x2, RSMI_PWR_PROF_PRST_POWER_SAVING_MASK = 0x4, RSMI_PWR_PROF_PRST_COMPUTE_MASK = 0x8, RSMI_PWR_PROF_PRST_VR_MASK = 0x10, **RSMI_PWR_PROF_PRST_3D_FULL_SCR_MASK** = 0x20, RSMI_PWR_PROF_PRST_BOOTUP_DEFAULT = 0x40, RSMI_PWR_PROF_PRST_LAST = RSMI_PWR_PROF_PRST_BOOTUP_DEFAULT, **RSMI_PWR_PROF_PRST_INVALID** = 0xFFFFFFFFFFFFFFFF }

  Pre-set Profile Selections. These bitmasks can be AND'd with the rsmi_power_profile_status_t.available_profiles returned from rsmi_dev_power_profile_presets_get to determine which power profiles are supported by the system.

- enum rsmi_gpu_block_t { RSMI_GPU_BLOCK_INVALID = 0x0000000000000000, **RSMI_GPU_BLOCK_FIRST** = 0x0000000000000001, RSMI_GPU_BLOCK_UMC = RSMI_GPU_BLOCK_FIRST, RSMI_GPU_BLOCK_SDMA = 0x0000000000000002, RSMI_GPU_BLOCK_GFX = 0x0000000000000004, RSMI_GPU_BLOCK_MMHUB = 0x0000000000000008, RSMI_GPU_BLOCK_ATHUB = 0x0000000000000010, RSMI_GPU_BLOCK_PCIE_BIF = 0x0000000000000020, RSMI_GPU_BLOCK_HDP = 0x0000000000000040, RSMI_GPU_BLOCK_XGMI_WAFL = 0x0000000000000080, RSMI_GPU_BLOCK_DF = 0x0000000000000100, RSMI_GPU_BLOCK_SMN = 0x0000000000000200, RSMI_GPU_BLOCK_SEM = 0x0000000000000400, RSMI_GPU_BLOCK_MP0 = 0x0000000000000800, RSMI_GPU_BLOCK_MP1 = 0x0000000000001000, RSMI_GPU_BLOCK_FUSE = 0x0000000000002000, RSMI_GPU_BLOCK_LAST = RSMI_GPU_BLOCK_FUSE, **RSMI_GPU_BLOCK_RESERVED** = 0x8000000000000000 }

  This enum is used to identify different GPU blocks.

- enum rsmi_ras_err_state_t { RSMI_RAS_ERR_STATE_NONE = 0, RSMI_RAS_ERR_STATE_DISABLED, RSMI_RAS_ERR_STATE_PARITY, RSMI_RAS_ERR_STATE_SING_C, RSMI_RAS_ERR_STATE_MULT_UC, RSMI_RAS_ERR_STATE_POISON, RSMI_RAS_ERR_STATE_ENABLED, **RSMI_RAS_ERR_STATE_LAST** = RSMI_RAS_ERR_STATE_ENABLED, **RSMI_RAS_ERR_STATE_INVALID** = 0xFFFFFFFF }

  The current ECC state.

- enum rsmi_memory_type_t { **RSMI_MEM_TYPE_FIRST** = 0, RSMI_MEM_TYPE_VRAM = RSMI_MEM_TYPE_FIRST, RSMI_MEM_TYPE_VIS_VRAM, RSMI_MEM_TYPE_GTT, **RSMI_MEM_TYPE_LAST** = RSMI_MEM_TYPE_GTT }

  Types of memory.

- enum rsmi_freq_ind_t { RSMI_FREQ_IND_MIN = 0, RSMI_FREQ_IND_MAX = 1, RSMI_FREQ_IND_INVALID = 0xFFFFFFFF }

  The values of this enum are used as frequency identifiers.

- enum rsmi_fw_block_t { **RSMI_FW_BLOCK_FIRST** = 0, **RSMI_FW_BLOCK_ASD** = RSMI_FW_BLOCK_FIRST, **RSMI_FW_BLOCK_CE**, **RSMI_FW_BLOCK_DMCU**, **RSMI_FW_BLOCK_MC**, **RSMI_FW_BLOCK_ME**, **RSMI_FW_BLOCK_MEC**, **RSMI_FW_BLOCK_MEC2**, **RSMI_FW_BLOCK_PFP**, **RSMI_FW_BLOCK_RLC**, **RSMI_FW_BLOCK_RLC_SRLC**, **RSMI_FW_BLOCK_RLC_SRLG**, **RSMI_FW_BLOCK_RLC_SRLS**, **RSMI_FW_BLOCK_SDMA**, **RSMI_FW_BLOCK_SDMA2**, **RSMI_FW_BLOCK_SMC**, **RSMI_FW_BLOCK_SOS**, **RSMI_FW_BLOCK_TA_RAS**, **RSMI_FW_BLOCK_TA_XGMI**, **RSMI_FW_BLOCK_UVD**, **RSMI_FW_BLOCK_VCE**, **RSMI_FW_BLOCK_VCN**, **RSMI_FW_BLOCK_LAST** = RSMI_FW_BLOCK_VCN }

  The values of this enum are used to identify the various firmware blocks.

- enum rsmi_xgmi_status_t { **RSMI_XGMI_STATUS_NO_ERRORS** = 0, **RSMI_XGMI_STATUS_ERROR**, **RSMI_XGMI_STATUS_MULTIPLE_ERRORS** }

XGMI Status.

- enum rsmi_memory_page_status_t { RSMI_MEM_PAGE_STATUS_RESERVED = 0, RSMI_MEM_PAGE_STATUS_PENDING RSMI_MEM_PAGE_STATUS_UNRESERVABLE }

  Reserved Memory Page States.

- enum _RSMI_IO_LINK_TYPE { RSMI_IOLINK_TYPE_UNDEFINED = 0, RSMI_IOLINK_TYPE_PCIEXPRESS = 1, RSMI_IOLINK_TYPE_XGMI = 2, RSMI_IOLINK_TYPE_NUMIOLINKTYPES, RSMI_IOLINK_TYPE_SIZE = 0xFFFFFFFF }

  Types for IO Link.

Functions

- rsmi_status_t rsmi_init (uint64_t init_flags)

  Initialize ROCm SMI.

- rsmi_status_t rsmi_shut_down (void)

  Shutdown ROCm SMI.

- rsmi_status_t rsmi_num_monitor_devices (uint32_t *num_devices)

  Get the number of devices that have monitor information.

- rsmi_status_t rsmi_dev_id_get (uint32_t dv_ind, uint16_t *id)

  Get the device id associated with the device with provided device index.

- rsmi_status_t rsmi_dev_vendor_id_get (uint32_t dv_ind, uint16_t *id)

  Get the device vendor id associated with the device with provided device index.

- rsmi_status_t rsmi_dev_name_get (uint32_t dv_ind, char *name, size_t len)

  Get the name string of a gpu device.

- rsmi_status_t rsmi_dev_brand_get (uint32_t dv_ind, char *brand, uint32_t len)

  Get the brand string of a gpu device.

- rsmi_status_t rsmi_dev_vendor_name_get (uint32_t dv_ind, char *name, size_t len)

  Get the name string for a give vendor ID.

- rsmi_status_t rsmi_dev_vram_vendor_get (uint32_t dv_ind, char *brand, uint32_t len)

  Get the vram vendor string of a gpu device.

- rsmi_status_t rsmi_dev_serial_number_get (uint32_t dv_ind, char *serial_num, uint32_t len)

  Get the serial number string for a device.

- rsmi_status_t rsmi_dev_subsystem_id_get (uint32_t dv_ind, uint16_t *id)

  Get the subsystem device id associated with the device with provided device index.

- rsmi_status_t rsmi_dev_subsystem_name_get (uint32_t dv_ind, char *name, size_t len)

  Get the name string for the device subsytem.

- rsmi_status_t rsmi_dev_drm_render_minor_get (uint32_t dv_ind, uint32_t *minor)

  Get the drm minor number associated with this device.

- rsmi_status_t rsmi_dev_subsystem_vendor_id_get (uint32_t dv_ind, uint16_t *id)

  Get the device subsystem vendor id associated with the device with provided device index.

- rsmi_status_t rsmi_dev_unique_id_get (uint32_t dv_ind, uint64_t *id)

  Get Unique ID.

- rsmi_status_t rsmi_dev_pci_bandwidth_get (uint32_t dv_ind, rsmi_pcie_bandwidth_t *bandwidth)

  Get the list of possible PCIe bandwidths that are available.

- rsmi_status_t rsmi_dev_pci_id_get (uint32_t dv_ind, uint64_t *bdfid)

  Get the unique PCI device identifier associated for a device.

- rsmi_status_t rsmi_topo_numa_affinity_get (uint32_t dv_ind, uint32_t *numa_node)

  Get the NUMA node associated with a device.

- rsmi_status_t rsmi_dev_pci_throughput_get (uint32_t dv_ind, uint64_t *sent, uint64_t *received, uint64_t *max_pkt_sz)

  Get PCIe traffic information.

- rsmi_status_t rsmi_dev_pci_replay_counter_get (uint32_t dv_ind, uint64_t *counter)

  Get PCIe replay counter.
- rsmi_status_t rsmi_dev_pci_bandwidth_set (uint32_t dv_ind, uint64_t bw_bitmask)

  Control the set of allowed PCIe bandwidths that can be used.
- rsmi_status_t rsmi_dev_power_ave_get (uint32_t dv_ind, uint32_t sensor_ind, uint64_t *power)

  Get the average power consumption of the device with provided device index.
- rsmi_status_t rsmi_dev_power_cap_get (uint32_t dv_ind, uint32_t sensor_ind, uint64_t *cap)

  Get the cap on power which, when reached, causes the system to take action to reduce power.
- rsmi_status_t rsmi_dev_power_cap_range_get (uint32_t dv_ind, uint32_t sensor_ind, uint64_t *max, uint64_t *min)

  Get the range of valid values for the power cap.
- rsmi_status_t rsmi_dev_power_cap_set (uint32_t dv_ind, uint32_t sensor_ind, uint64_t cap)

  Set the power cap value.
- rsmi_status_t rsmi_dev_power_profile_set (uint32_t dv_ind, uint32_t reserved, rsmi_power_profile_preset_masks_t profile)

  Set the power profile.
- rsmi_status_t rsmi_dev_memory_total_get (uint32_t dv_ind, rsmi_memory_type_t mem_type, uint64_t *total)

  Get the total amount of memory that exists.
- rsmi_status_t rsmi_dev_memory_usage_get (uint32_t dv_ind, rsmi_memory_type_t mem_type, uint64_t *used)

  Get the current memory usage.
- rsmi_status_t rsmi_dev_memory_busy_percent_get (uint32_t dv_ind, uint32_t *busy_percent)

  Get percentage of time any device memory is being used.
- rsmi_status_t rsmi_dev_memory_reserved_pages_get (uint32_t dv_ind, uint32_t *num_pages, rsmi_retired_page_record_t *records)

  Get information about reserved ("retired") memory pages.
- rsmi_status_t rsmi_dev_fan_rpms_get (uint32_t dv_ind, uint32_t sensor_ind, int64_t *speed)

  Get the fan speed in RPMs of the device with the specified device index and 0-based sensor index.
- rsmi_status_t rsmi_dev_fan_speed_get (uint32_t dv_ind, uint32_t sensor_ind, int64_t *speed)

  Get the fan speed for the specified device as a value relative to RSMI_MAX_FAN_SPEED.
- rsmi_status_t rsmi_dev_fan_speed_max_get (uint32_t dv_ind, uint32_t sensor_ind, uint64_t *max_speed)

  Get the max. fan speed of the device with provided device index.
- rsmi_status_t rsmi_dev_temp_metric_get (uint32_t dv_ind, uint32_t sensor_type, rsmi_temperature_metric_t metric, int64_t *temperature)

  Get the temperature metric value for the specified metric, from the specified temperature sensor on the specified device.
- rsmi_status_t rsmi_dev_volt_metric_get (uint32_t dv_ind, rsmi_voltage_type_t sensor_type, rsmi_voltage_metric_t metric, int64_t *voltage)

  Get the voltage metric value for the specified metric, from the specified voltage sensor on the specified device.
- rsmi_status_t rsmi_dev_fan_reset (uint32_t dv_ind, uint32_t sensor_ind)

  Reset the fan to automatic driver control.
- rsmi_status_t rsmi_dev_fan_speed_set (uint32_t dv_ind, uint32_t sensor_ind, uint64_t speed)

  Set the fan speed for the specified device with the provided speed, in RPMs.
- rsmi_status_t rsmi_dev_busy_percent_get (uint32_t dv_ind, uint32_t *busy_percent)

  Get percentage of time device is busy doing any processing.
- rsmi_status_t rsmi_dev_perf_level_get (uint32_t dv_ind, rsmi_dev_perf_level_t *perf)

  Get the performance level of the device with provided device index.
- rsmi_status_t rsmi_dev_overdrive_level_get (uint32_t dv_ind, uint32_t *od)

  Get the overdrive percent associated with the device with provided device index.
- rsmi_status_t rsmi_dev_gpu_clk_freq_get (uint32_t dv_ind, rsmi_clk_type_t clk_type, rsmi_frequencies_t *f)

  Get the list of possible system clock speeds of device for a specified clock type.

- rsmi_status_t rsmi_dev_od_volt_info_get (uint32_t dv_ind, rsmi_od_volt_freq_data_t *odv)

  This function retrieves the voltage/frequency curve information.
- rsmi_status_t rsmi_dev_od_volt_curve_regions_get (uint32_t dv_ind, uint32_t *num_regions, rsmi_freq_volt_region_t *buffer)

  This function will retrieve the current valid regions in the frequency/voltage space.
- rsmi_status_t rsmi_dev_power_profile_presets_get (uint32_t dv_ind, uint32_t sensor_ind, rsmi_power_profile_status_t *status)

  Get the list of available preset power profiles and an indication of which profile is currently active.
- rsmi_status_t rsmi_dev_perf_level_set (int32_t dv_ind, rsmi_dev_perf_level_t perf_lvl)

  Set the PowerPlay performance level associated with the device with provided device index with the provided value.
- rsmi_status_t rsmi_dev_overdrive_level_set (int32_t dv_ind, uint32_t od)

  Set the overdrive percent associated with the device with provided device index with the provided value. See details for WARNING.
- rsmi_status_t rsmi_dev_gpu_clk_freq_set (uint32_t dv_ind, rsmi_clk_type_t clk_type, uint64_t freq_bitmask)

  Control the set of allowed frequencies that can be used for the specified clock.
- rsmi_status_t rsmi_version_get (rsmi_version_t *version)

  Get the build version information for the currently running build of RSMI.
- rsmi_status_t rsmi_version_str_get (rsmi_sw_component_t component, char *ver_str, uint32_t len)

  Get the driver version string for the current system.
- rsmi_status_t rsmi_dev_vbios_version_get (uint32_t dv_ind, char *vbios, uint32_t len)

  Get the VBIOS identifer string.
- rsmi_status_t rsmi_dev_firmware_version_get (uint32_t dv_ind, rsmi_fw_block_t block, uint64_t *fw_version)

  Get the firmware versions for a device.
- rsmi_status_t rsmi_dev_ecc_count_get (uint32_t dv_ind, rsmi_gpu_block_t block, rsmi_error_count_t *ec)

  Retrieve the error counts for a GPU block.
- rsmi_status_t rsmi_dev_ecc_enabled_get (uint32_t dv_ind, uint64_t *enabled_blocks)

  Retrieve the enabled ECC bit-mask.
- rsmi_status_t rsmi_dev_ecc_status_get (uint32_t dv_ind, rsmi_gpu_block_t block, rsmi_ras_err_state_t *state)

  Retrieve the ECC status for a GPU block.
- rsmi_status_t rsmi_status_string (rsmi_status_t status, const char **status_string)

  Get a description of a provided RSMI error status.
- rsmi_status_t rsmi_dev_counter_group_supported (uint32_t dv_ind, rsmi_event_group_t group)

  Tell if an event group is supported by a given device.
- rsmi_status_t rsmi_dev_counter_create (uint32_t dv_ind, rsmi_event_type_t type, rsmi_event_handle_t *evnt_handle)

  Create a performance counter object.
- rsmi_status_t rsmi_dev_counter_destroy (rsmi_event_handle_t evnt_handle)

  Deallocate a performance counter object.
- rsmi_status_t rsmi_counter_control (rsmi_event_handle_t evt_handle, rsmi_counter_command_t cmd, void *cmd_args)

  Issue performance counter control commands.
- rsmi_status_t rsmi_counter_read (rsmi_event_handle_t evt_handle, rsmi_counter_value_t *value)

  Read the current value of a performance counter.
- rsmi_status_t rsmi_counter_available_counters_get (uint32_t dv_ind, rsmi_event_group_t grp, uint32_t *available)

  Get the number of currently available counters.
- rsmi_status_t rsmi_compute_process_info_get (rsmi_process_info_t *procs, uint32_t *num_items)

  Get process information about processes currently using GPU.
- rsmi_status_t rsmi_compute_process_info_by_pid_get (uint32_t pid, rsmi_process_info_t *proc)

  Get process information about a specific process.
- rsmi_status_t rsmi_compute_process_gpus_get (uint32_t pid, uint32_t *dv_indices, uint32_t *num_devices)

Get the device indices currently being used by a process.

- rsmi_status_t rsmi_dev_xgmi_error_status (uint32_t dv_ind, rsmi_xgmi_status_t *status)

  Retrieve the XGMI error status for a device.
- rsmi_status_t rsmi_dev_xgmi_error_reset (uint32_t dv_ind)

  Reset the XGMI error status for a device.
- rsmi_status_t rsmi_dev_xgmi_hive_id_get (uint32_t dv_ind, uint64_t *hive_id)

  Retrieve the XGMI hive id for a device.
- rsmi_status_t rsmi_topo_get_numa_node_number (uint32_t dv_ind, uint32_t *numa_node)

  Retrieve the NUMA CPU node number for a device.
- rsmi_status_t rsmi_topo_get_link_weight (uint32_t dv_ind_src, uint32_t dv_ind_dst, uint64_t *weight)

  Retrieve the weight for a connection between 2 GPUs.
- rsmi_status_t rsmi_topo_get_link_type (uint32_t dv_ind_src, uint32_t dv_ind_dst, uint64_t *hops, RSMI_IO_LINK_TYPE *type)

  Retrieve the hops and the connection type between 2 GPUs.
- rsmi_status_t rsmi_dev_supported_func_iterator_open (uint32_t dv_ind, rsmi_func_id_iter_handle_t *handle)

  Get a function name iterator of supported RSMI functions for a device.
- rsmi_status_t rsmi_dev_supported_variant_iterator_open (rsmi_func_id_iter_handle_t obj_h, rsmi_func_id_iter_handle_t *var_iter)

  Get a variant iterator for a given handle.
- rsmi_status_t rsmi_func_iter_next (rsmi_func_id_iter_handle_t handle)

  Advance a function identifer iterator.
- rsmi_status_t rsmi_dev_supported_func_iterator_close (rsmi_func_id_iter_handle_t *handle)

  Close a variant iterator handle.
- rsmi_status_t rsmi_func_iter_value_get (rsmi_func_id_iter_handle_t handle, rsmi_func_id_value_t *value)

  Get the value associated with a function/variant iterator.
- rsmi_status_t rsmi_event_notification_init (uint32_t dv_ind)

  Prepare to collect event notifications for a GPU.
- rsmi_status_t rsmi_event_notification_mask_set (uint32_t dv_ind, uint64_t mask)

  Specify which events to collect for a device.
- rsmi_status_t rsmi_event_notification_get (int timeout_ms, uint32_t *num_elem, rsmi_evt_notification_data_t *data)

  Collect event notifications, waiting a specified amount of time.
- rsmi_status_t rsmi_event_notification_stop (uint32_t dv_ind)

  Close any file handles and free any resources used by event notification for a GPU.

7.1.1 Detailed Description

The rocm_smi library api is new, and therefore subject to change either at the ABI or API level. Instead of marking every function prototype as "unstable", we are instead saying the API is unstable (i.e., changes are possible) while the major version remains 0. This means that if the API/ABI changes, we will not increment the major version to 1. Once the ABI stabilizes, we will increment the major version to 1, and thereafter increment it on all ABI breaks.

Main header file for the ROCm SMI library. All required function, structure, enum, etc. definitions should be defined in this file.

7.1.2 Macro Definition Documentation

7.1.2.1 RSMI_MAX_FAN_SPEED

```
#define RSMI_MAX_FAN_SPEED 255
```

Maximum possible value for fan speed. Should be used as the denominator when determining fan speed percentage.

7.1.2.2 RSMI_DEFAULT_VARIANT

```
#define RSMI_DEFAULT_VARIANT 0xFFFFFFFFFFFFFFFF
```

Place-holder "variant" for functions that have don't have any variants, but do have monitors or sensors.

7.1.3 Typedef Documentation

7.1.3.1 rsmi_event_handle_t

```
typedef uintptr_t rsmi_event_handle_t
```

Handle to performance event counter.

Event counter types

7.1.4 Enumeration Type Documentation

7.1.4.1 rsmi_status_t

```
enum rsmi_status_t
```

Error codes retured by rocm_smi_lib functions.

Enumerator

RSMI_STATUS_SUCCESS	Operation was successful.
RSMI_STATUS_INVALID_ARGS	Passed in arguments are not valid.
RSMI_STATUS_NOT_SUPPORTED	The requested information or action is not available for the given input, on the given system
RSMI_STATUS_FILE_ERROR	Problem accessing a file. This may because the operation is not supported by the Linux kernel version running on the executing machine
RSMI_STATUS_PERMISSION	Permission denied/EACCESS file error. Many functions require root access to run.
RSMI_STATUS_OUT_OF_RESOURCES	Unable to acquire memory or other resource
RSMI_STATUS_INTERNAL_EXCEPTION	An internal exception was caught.

Enumerator

RSMI_STATUS_INPUT_OUT_OF_BOUNDS	The provided input is out of allowable or safe range
RSMI_STATUS_INIT_ERROR	An error occurred when rsmi initializing internal data structures
RSMI_STATUS_NOT_YET_IMPLEMENTED	The requested function has not yet been implemented in the current system for the current devices
RSMI_STATUS_NOT_FOUND	An item was searched for but not found
RSMI_STATUS_INSUFFICIENT_SIZE	Not enough resources were available for the operation
RSMI_STATUS_INTERRUPT	An interrupt occurred during execution of function
RSMI_STATUS_UNEXPECTED_SIZE	An unexpected amount of data was read
RSMI_STATUS_NO_DATA	No data was found for a given input
RSMI_STATUS_UNEXPECTED_DATA	The data read or provided to function is not what was expected
RSMI_STATUS_BUSY	A resource or mutex could not be acquired because it is already being used
RSMI_STATUS_REFCOUNT_OVERFLOW	exceeded INT32_MAX An internal reference counter
RSMI_STATUS_UNKNOWN_ERROR	An unknown error occurred.

7.1.4.2 rsmi_init_flags_t

`enum rsmi_init_flags_t`

Initialization flags.

Initialization flags may be OR'd together and passed to rsmi_init().

Enumerator

RSMI_INIT_FLAG_ALL_GPUS	Attempt to add all GPUs found (including non-AMD) to the list of devices from which SMI information can be retrieved. By default, only AMD devices are ennumerated by RSMI.
RSMI_INIT_FLAG_RESRV_TEST1	Reserved for test.

7.1.4.3 rsmi_dev_perf_level_t

`enum rsmi_dev_perf_level_t`

PowerPlay performance levels.

Enumerator

RSMI_DEV_PERF_LEVEL_AUTO	Performance level is "auto".
RSMI_DEV_PERF_LEVEL_LOW	Keep PowerPlay levels "low", regardless of workload
RSMI_DEV_PERF_LEVEL_HIGH	Keep PowerPlay levels "high", regardless of workload

Enumerator

RSMI_DEV_PERF_LEVEL_MANUAL	Only use values defined by manually setting the RSMI_CLK_TYPE_SYS speed
RSMI_DEV_PERF_LEVEL_STABLE_STD	Stable power state with profiling clocks
RSMI_DEV_PERF_LEVEL_STABLE_PEAK	Stable power state with peak clocks.
RSMI_DEV_PERF_LEVEL_STABLE_MIN_MCLK	Stable power state with minimum memory clock
RSMI_DEV_PERF_LEVEL_STABLE_MIN_SCLK	Stable power state with minimum system clock
RSMI_DEV_PERF_LEVEL_UNKNOWN	Unknown performance level.

7.1.4.4 rsmi_sw_component_t

`enum rsmi_sw_component_t`

Available clock types.

Software components

Enumerator

RSMI_SW_COMP_DRIVER	Driver.

7.1.4.5 rsmi_event_group_t

`enum rsmi_event_group_t`

Enum denoting an event group. The value of the enum is the base value for all the event enums in the group.

Event Groups

Enumerator

RSMI_EVNT_GRP_XGMI	Data Fabric (XGMI) related events.

7.1.4.6 rsmi_event_type_t

`enum rsmi_event_type_t`

Event type enum. Events belonging to a particular event group rsmi_event_group_t should begin ennumerating at the rsmi_event_group_t value for that group.

Event types

Enumerator

RSMI_EVNT_XGMI_0_NOP_TX	NOPs sent to neighbor 0.
RSMI_EVNT_XGMI_0_REQUEST_TX	Outgoing requests to neighbor 0
RSMI_EVNT_XGMI_0_RESPONSE_TX	Outgoing responses to neighbor 0
RSMI_EVNT_XGMI_0_BEATS_TX	Data beats sent to neighbor 0; Each beat represents 32 bytes. XGMI throughput can be calculated by multiplying a BEATs event such as RSMI_EVNT_XGMI_0_BEATS_TX by 32 and dividing by the time for which event collection occurred, rsmi_counter_value_t.time_running (which is in nanoseconds). To get bytes per second, multiply this value by 10^9. Throughput = BEATS/time_running $* 10^9$ (bytes/second)
RSMI_EVNT_XGMI_1_NOP_TX	NOPs sent to neighbor 1.
RSMI_EVNT_XGMI_1_REQUEST_TX	neighbor 1 Outgoing requests to
RSMI_EVNT_XGMI_1_RESPONSE_TX	Outgoing responses to neighbor 1
RSMI_EVNT_XGMI_1_BEATS_TX	Data beats sent to neighbor 1; Each beat represents 32 bytes

7.1.4.7 rsmi_counter_command_t

`enum rsmi_counter_command_t`

Event counter commands

Enumerator

RSMI_CNTR_CMD_START	Start the counter.
RSMI_CNTR_CMD_STOP	Stop the counter; note that this should not be used before reading. It is for temporarily disabling the counter.

7.1.4.8 rsmi_evt_notification_type_t

`enum rsmi_evt_notification_type_t`

Event notification event types

Enumerator

RSMI_EVT_NOTIF_VMFAULT	VM page fault.

7.1.4.9 rsmi_clk_type_t

enum rsmi_clk_type_t

Clock types

Enumerator

RSMI_CLK_TYPE_SYS	System clock.
RSMI_CLK_TYPE_DF	Data Fabric clock (for ASICs running on a separate clock)
RSMI_CLK_TYPE_DCEF	Display Controller Engine clock.
RSMI_CLK_TYPE_SOC	SOC clock.
RSMI_CLK_TYPE_MEM	Memory clock.

7.1.4.10 rsmi_temperature_metric_t

enum rsmi_temperature_metric_t

Temperature Metrics. This enum is used to identify various temperature metrics. Corresponding values will be in millidegress Celcius.

Enumerator

RSMI_TEMP_CURRENT	Temperature current value.
RSMI_TEMP_MAX	Temperature max value.
RSMI_TEMP_MIN	Temperature min value.
RSMI_TEMP_MAX_HYST	Temperature hysteresis value for max limit. (This is an absolute temperature, not a delta).
RSMI_TEMP_MIN_HYST	Temperature hysteresis value for min limit. (This is an absolute temperature, not a delta).
RSMI_TEMP_CRITICAL	Temperature critical max value, typically greater than corresponding temp_max values.
RSMI_TEMP_CRITICAL_HYST	Temperature hysteresis value for critical limit. (This is an absolute temperature, not a delta).
RSMI_TEMP_EMERGENCY	Temperature emergency max value, for chips supporting more than two upper temperature limits. Must be equal or greater than corresponding temp_crit values.
RSMI_TEMP_EMERGENCY_HYST	Temperature hysteresis value for emergency limit. (This is an absolute temperature, not a delta).
RSMI_TEMP_CRIT_MIN	Temperature critical min value, typically lower than corresponding temperature minimum values.
RSMI_TEMP_CRIT_MIN_HYST	Temperature hysteresis value for critical minimum limit. (This is an absolute temperature, not a delta).
RSMI_TEMP_OFFSET	Temperature offset which is added to the temperature reading by the chip.
RSMI_TEMP_LOWEST	Historical minimum temperature.
RSMI_TEMP_HIGHEST	Historical maximum temperature.

7.1.4.11 rsmi_temperature_type_t

enum rsmi_temperature_type_t

This ennumeration is used to indicate from which part of the device a temperature reading should be obtained.

Enumerator

RSMI_TEMP_TYPE_EDGE	Edge GPU temperature.
RSMI_TEMP_TYPE_JUNCTION	Junction/hotspot temperature
RSMI_TEMP_TYPE_MEMORY	VRAM temperature.
RSMI_TEMP_TYPE_INVALID	Invalid type.

7.1.4.12 rsmi_voltage_metric_t

enum rsmi_voltage_metric_t

Voltage Metrics. This enum is used to identify various Volatge metrics. Corresponding values will be in millivolt.

Enumerator

RSMI_VOLT_CURRENT	Voltage current value.
RSMI_VOLT_MAX	Voltage max value.
RSMI_VOLT_MIN_CRIT	Voltage critical min value.
RSMI_VOLT_MIN	Voltage min value.
RSMI_VOLT_MAX_CRIT	Voltage critical max value.
RSMI_VOLT_AVERAGE	Average voltage.
RSMI_VOLT_LOWEST	Historical minimum voltage.
RSMI_VOLT_HIGHEST	Historical maximum voltage.

7.1.4.13 rsmi_voltage_type_t

enum rsmi_voltage_type_t

This ennumeration is used to indicate which type of voltage reading should be obtained.

Enumerator

RSMI_VOLT_TYPE_VDDGFX	Vddgfx GPU voltage
RSMI_VOLT_TYPE_INVALID	Invalid type.

7.1.4.14 rsmi_power_profile_preset_masks_t

enum rsmi_power_profile_preset_masks_t

Pre-set Profile Selections. These bitmasks can be AND'd with the rsmi_power_profile_status_t.available_profiles returned from rsmi_dev_power_profile_presets_get to determine which power profiles are supported by the system.

Enumerator

RSMI_PWR_PROF_PRST_CUSTOM_MASK	Custom Power Profile.
RSMI_PWR_PROF_PRST_VIDEO_MASK	Video Power Profile.
RSMI_PWR_PROF_PRST_POWER_SAVING_MASK	Power Saving Profile.
RSMI_PWR_PROF_PRST_COMPUTE_MASK	Compute Saving Profile.
RSMI_PWR_PROF_PRST_VR_MASK	VR Power Profile. 3D Full Screen Power Profile
RSMI_PWR_PROF_PRST_BOOTUP_DEFAULT	Default Boot Up Profile.
RSMI_PWR_PROF_PRST_LAST	Invalid power profile.

7.1.4.15 rsmi_gpu_block_t

enum rsmi_gpu_block_t

This enum is used to identify different GPU blocks.

Enumerator

RSMI_GPU_BLOCK_INVALID	Used to indicate an invalid block
RSMI_GPU_BLOCK_UMC	UMC block.
RSMI_GPU_BLOCK_SDMA	SDMA block.
RSMI_GPU_BLOCK_GFX	GFX block.
RSMI_GPU_BLOCK_MMHUB	MMHUB block.
RSMI_GPU_BLOCK_ATHUB	ATHUB block.
RSMI_GPU_BLOCK_PCIE_BIF	PCIE_BIF block.
RSMI_GPU_BLOCK_HDP	HDP block.
RSMI_GPU_BLOCK_XGMI_WAFL	XGMI block.
RSMI_GPU_BLOCK_DF	DF block.
RSMI_GPU_BLOCK_SMN	SMN block.
RSMI_GPU_BLOCK_SEM	SEM block.
RSMI_GPU_BLOCK_MP0	MP0 block.
RSMI_GPU_BLOCK_MP1	MP1 block.
RSMI_GPU_BLOCK_FUSE	Fuse block.
RSMI_GPU_BLOCK_LAST	for supported blocks The highest bit position

7.1.4.16 rsmi_ras_err_state_t

```
enum rsmi_ras_err_state_t
```

The current ECC state.

Enumerator

RSMI_RAS_ERR_STATE_NONE	No current errors.
RSMI_RAS_ERR_STATE_DISABLED	ECC is disabled.
RSMI_RAS_ERR_STATE_PARITY	ECC errors present, but type unknown.
RSMI_RAS_ERR_STATE_SING_C	Single correctable error.
RSMI_RAS_ERR_STATE_MULT_UC	Multiple uncorrectable errors.
RSMI_RAS_ERR_STATE_POISON	Firmware detected error and isolated page. Treat as uncorrectable.
RSMI_RAS_ERR_STATE_ENABLED	ECC is enabled.

7.1.4.17 rsmi_memory_type_t

```
enum rsmi_memory_type_t
```

Types of memory.

Enumerator

RSMI_MEM_TYPE_VRAM	VRAM memory.
RSMI_MEM_TYPE_VIS_VRAM	VRAM memory that is visible.
RSMI_MEM_TYPE_GTT	GTT memory.

7.1.4.18 rsmi_freq_ind_t

```
enum rsmi_freq_ind_t
```

The values of this enum are used as frequency identifiers.

Enumerator

RSMI_FREQ_IND_MIN	Index used for the minimum frequency value.
RSMI_FREQ_IND_MAX	Index used for the maximum frequency value.
RSMI_FREQ_IND_INVALID	An invalid frequency index.

7.1.4.19 rsmi_memory_page_status_t

```
enum rsmi_memory_page_status_t
```

Reserved Memory Page States.

Enumerator

RSMI_MEM_PAGE_STATUS_RESERVED	Reserved. This gpu page is reserved and not available for use
RSMI_MEM_PAGE_STATUS_PENDING	Pending. This gpu page is marked as bad and will be marked reserved at the next window.
RSMI_MEM_PAGE_STATUS_UNRESERVABLE	Unable to reserve this page.

7.1.4.20 _RSMI_IO_LINK_TYPE

```
enum _RSMI_IO_LINK_TYPE
```

Types for IO Link.

Enumerator

RSMI_IOLINK_TYPE_UNDEFINED	unknown type.
RSMI_IOLINK_TYPE_PCIEXPRESS	PCI Express.
RSMI_IOLINK_TYPE_XGMI	XGMI.
RSMI_IOLINK_TYPE_NUMIOLINKTYPES	Number of IO Link types.
RSMI_IOLINK_TYPE_SIZE	Max of IO Link types.

7.1.5 Function Documentation

7.1.5.1 rsmi_dev_volt_metric_get()

```
rsmi_status_t rsmi_dev_volt_metric_get (
            uint32_t dv_ind,
            rsmi_voltage_type_t sensor_type,
            rsmi_voltage_metric_t metric,
            int64_t * voltage )
```

Get the voltage metric value for the specified metric, from the specified voltage sensor on the specified device.

Given a device index `dv_ind`, a sensor type `sensor_type`, a rsmi_voltage_metric_t `metric` and a pointer to an int64_t `voltage`, this function will write the value of the metric indicated by `metric` and `sensor_type` to the memory location `voltage`.

Parameters

in	*dv_ind*	a device index
in	*sensor_type*	part of device from which voltage should be obtained. This should come from the enum rsmi_voltage_type_t
in	*metric*	enum indicated which voltage value should be retrieved
in,out	*voltage*	a pointer to int64_t to which the voltage will be written, in millivolts. If this parameter is nullptr, this function will return RSMI_STATUS_INVALID_ARGS if the function is supported with the provided, arguments and RSMI_STATUS_NOT_SUPPORTED if it is not supported with the provided arguments.

Return values

RSMI_STATUS_SUCCESS	call was successful
RSMI_STATUS_NOT_SUPPORTED	installed software or hardware does not support this function with the given arguments
RSMI_STATUS_INVALID_ARGS	the provided arguments are not valid

7.1.5.2 rsmi_event_notification_init()

```
rsmi_status_t rsmi_event_notification_init (
            uint32_t dv_ind )
```

Prepare to collect event notifications for a GPU.

This function prepares to collect events for the GPU with device ID `dv_ind`, by initializing any required system parameters. This call may open files which will remain open until rsmi_event_notification_stop() is called.

Parameters

dv_ind	a device index corresponding to the device on which to listen for events

Return values

RSMI_STATUS_SUCCESS	is returned upon successful call.

7.1.5.3 rsmi_event_notification_mask_set()

```
rsmi_status_t rsmi_event_notification_mask_set (
            uint32_t dv_ind,
            uint64_t mask )
```

Specify which events to collect for a device.

Given a device index `dv_ind` and a `mask` consisting of elements of rsmi_evt_notification_type_t OR'd together, this function will listen for the events specified in `mask` on the device corresponding to `dv_ind`.

Parameters

dv_ind	a device index corresponding to the device on which to listen for events
mask	0 or more elements of rsmi_evt_notification_type_t OR'd together that indicate which event types to listen for.

Return values

RSMI_STATUS_INIT_ERROR	is returned if rsmi_event_notification_init() has not been called before a call to this function
RSMI_STATUS_SUCCESS	is returned upon successful call

7.1.5.4 rsmi_event_notification_get()

```
rsmi_status_t rsmi_event_notification_get (
            int timeout_ms,
            uint32_t * num_elem,
            rsmi_evt_notification_data_t * data )
```

Collect event notifications, waiting a specified amount of time.

Given a time period `timeout_ms` in milliseconds and a caller- provided buffer of rsmi_evt_notification_data_t's `data` with a length (in rsmi_evt_notification_data_t's, also specified by the caller) in the memory location pointed to by `num_elem`, this function will collect rsmi_evt_notification_type_t events for up to `timeout_ms` milliseconds, and write up to `*num_elem` event items to `data`. Upon return `num_elem` is updated with the number of events that were actually written. If events are already present when this function is called, it will write the events to the buffer then poll for new events if there is still caller-provided buffer available to write any new events that would be found.

This function requires prior calls to rsmi_event_notification_init() and rsmi_event_notification_mask_set(). This function polls for the occurrance of the events on the respective devices that were previously specified by rsmi_event_notification_mask_set().

Parameters

`in`	*timeout_ms*	number of milliseconds to wait for an event to occur
`in,out`	*num_elem*	pointer to uint32_t, provided by the caller. On input, this value tells how many rsmi_evt_notification_data_t elements are being provided by the caller with `data`. On output, the location pointed to by `num_elem` will contain the number of items written to the provided buffer.
`out`	*data*	pointer to a caller-provided memory buffer of size `num_elem` rsmi_evt_notification_data_t to which this function may safely write. If there are events found, up to `num_elem` event items will be written to `data`.

Return values

RSMI_STATUS_SUCCESS	The function ran successfully. The events that were found are written to `data` and `num_elems` is updated with the number of elements that were written.
RSMI_STATUS_NO_DATA	No events were found to collect.

7.1.5.5 rsmi_event_notification_stop()

```
rsmi_status_t rsmi_event_notification_stop (
            uint32_t dv_ind )
```

Close any file handles and free any resources used by event notification for a GPU.

Any resources used by event notification for the GPU with device index `dv_ind` will be free with this function. This includes freeing any memory and closing file handles. This should be called for every call to rsmi_event_notification_init()

Parameters

in	*dv_ind*	The device index of the GPU for which event notification resources will be free

Return values

RSMI_STATUS_INVALID_ARGS	resources for the given device have either already been freed, or were never allocated by rsmi_event_notification_init()
RSMI_STATUS_SUCCESS	is returned upon successful call

Index