# ROCmSMI

Generated by Doxygen 1.8.11

# Contents

# 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.

## 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.

## Building ROCm SMI

### 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 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,

```
$ mk -p build

$ cd build

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

$ make
```

The built library will appear in the `build` folder.

#### Building the Documentation

The documentation PDF file can be built with the following steps (continued from the steps above):

```
$ make doc

$ cd latex

$ make
```

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

#### 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=<location of ROCm SMI library>

$ mkdir <location for test build>

$ cd <location for test build>

$ cmake -DROCM_DIR=<location of ROCM SMI library .so> <ROCm SMI source root>/tests/rocm
_smi_test
```

To run the test, execute the program `rsmitst` that is built from the steps above. Make sure ROCm SMI library is in your library search path when executing the test program.

## Usage Basics

### 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.

## 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:

```
1 #include <stdint.h>
2 #include "rocm_smi/rocm_smi.h"
3 int main() {
4   rsmi_status_t ret;
5   uint32_t num_devices;
6   uint64_t dev_id;
7
8   // We will skip return code checks for this example, but it
9   // is recommended to always check this as some calls may not
10   // apply for some devices or ROCm releases
11
12   ret = rsmi_init(0);
13   ret = rsmi_num_monitor_devices(&num_devices);
14
15   for (int i=0; i < num_devices; ++i) {
16     ret = rsmi_dev_id_get(i, &dev_id);
17     // dev_id holds the device ID of device i, upon a
18     // successful call
19   }
20   ret = rsmi_shut_down();
21   return 0;
22 }
```

# 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

### 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_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_status_t rsmi_shut_down ( void )

Shutdown ROCm SMI.

Do any necessary clean up.

## 5.2 Identifier Queries

### 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_vendor_name_get (uint32_t id, char *name, size_t len)
  *Get the name string for a give vendor ID.*
- 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_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_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_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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.2.2.3 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.2.2.4 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 |
| `in` | *len* | the length of the caller provided buffer `name`. |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |
| *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_status_t rsmi_dev_vendor_name_get ( uint32_t *id*, char * *name*, size_t *len* )

Get the name string for a give vendor ID.

Given vendor ID `id`, 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` | *id* | a vendor ID |
| `in,out` | *name* | a pointer to a caller provided char buffer to which the name will be written |
| `in` | *len* | the length of the caller provided buffer `name`. |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |
| *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_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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.2.2.7 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 |
| `in` | *len* | the length of the caller provided buffer `name`. |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |
| *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. |

#### 5.2.2.8 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.2.2.9 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

## 5.3 PCIe Queries

### 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_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_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_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`.

**Parameters**

| | | |
|---|---|---|
| in | *dv_ind* | a device index |
| in,out | *bdfid* | a pointer to uint64_t to which the device bdfid value will be written |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.3.2.3 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. |

#### 5.3.2.4 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

## 5.4 PCIe Control

### 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_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. |

## 5.5 Power Queries

### 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_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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.5.2.2 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.5.2.3 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 |
| `in,out` | *min* | a pointer to a uint64_t that indicates the minimum possible power cap, in microwatts |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

## 5.6 Power Control

### 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 sensor_ind, 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_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,out` | *cap* | a uint64_t that indicates the desired power cap, in microwatts |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.6.2.2 rsmi_status_t rsmi_dev_power_profile_set ( uint32_t *dv_ind*, uint32_t *sensor_ind*, rsmi_power_profile_preset_masks_t *profile* )

Set the power profile.

Given a device index `dv_ind`, a sensor index sensor_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` | *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` | *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. |

## 5.7 Memory Queries

### 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.*

### 5.7.1 Detailed Description

These functions provide information about memory systems.

### 5.7.2 Function Documentation

#### 5.7.2.1 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.7.2.2 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 `total`.

**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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

## 5.8 Physical State Queries

### 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_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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.8.2.2 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.8.2.3 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.8.2.4 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. |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

## 5.9 Physical State Control

### 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_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. |

#### 5.9.2.2 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. |

## 5.10 Clock, Power and Performance Queries

### 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_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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call |

#### 5.10.2.2 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.10.2.3 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.10.2.4 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. |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.10.2.5 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` | *odv* | a pointer to an rsmi_od_volt_freq_data_t structure |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.10.2.6 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. |
| `in,out` | *buffer* | a caller provided buffer to which rsmi_freq_volt_region_t structures will be written |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.10.2.7 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

## 5.11 Clock, Power and Performance Control

### 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.*
- rsmi_status_t rsmi_dev_od_freq_range_set (uint32_t dv_ind, rsmi_clk_type_t clk, rsmi_range_t *range)

  *Set the frequency limits 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_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. |

#### 5.11.2.2 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 AND 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* | is returned upon successful call. |

#### 5.11.2.3 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. |

#### 5.11.2.4 rsmi_status_t rsmi_dev_od_freq_range_set ( uint32_t *dv_ind*, rsmi_clk_type_t *clk*, rsmi_range_t * *range* )

Set the frequency limits for the specified clock.

Given a device index `dv_ind`, a clock type (rsmi_clk_type_t) `clk`, and a pointer to a rsmi_range_t `range` containing the desired upper and lower frequency limits, this function will attempt to set the frequency limits to those specified in `range`.

**Parameters**

| | | |
|---|---|---|
| in | *dv_ind* | a device index |
| in | *clk* | The clock type for which the limits should be imposed. |
| in | *range* | A pointer to the rsmi_range_t containing the desired limits |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

## 5.12 Version Queries

### 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.*

### 5.12.1 Detailed Description

These functions provide version information about various subsystems.

### 5.12.2 Function Documentation

#### 5.12.2.1 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_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 VBIOS name will be written |
| `in` | *len* | The number of char's pointed to by `ver_str` which can safely be written to by this function. |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.12.2.3 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 |
| `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* | is returned upon successful call. |

## 5.13 Error Queries

### 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_mask)

  *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_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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.13.2.2 rsmi_status_t rsmi_dev_ecc_enabled_get ( uint32_t *dv_ind*, uint64_t * *enabled_mask* )

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 a bit_mask to memory pointed to by `enabled_mask`. Upon a successful call, the bitmask 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 the bits above RSMI_GPU_BLOCK_LAST correspond to blocks that do not yet have ECC support.

**Parameters**

| | | |
|---|---|---|
| in | *dv_ind* | a device index |
| in,out | *enabled_mask* | A pointer to a uint64_t to which the enabled mask will be written |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.13.2.3 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call. |

#### 5.13.2.4 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

### 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.

### 5.14.2 Function Documentation

#### 5.14.2.1 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 If the device does not support event group `group`

#### 5.14.2.2 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 type 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 |

**Return values**

| | |
|---|---|
| *RSMI_STATUS_SUCCESS* | is returned upon successful call |

#### 5.14.2.3 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 |

#### 5.14.2.4 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 |

#### 5.14.2.5 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 |

#### 5.14.2.6 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 |

# Chapter 6

# Data Structure Documentation

## 6.1 rsmi_counter_value_t Struct Reference

`#include <rocm_smi.h>`

### Data Fields

- uint64_t value
  *Counter value.*
- uint64_t time_enabled
  *Time that the counter was enabled.*
- uint64_t time_running
  *Time that che counter was running.*

### 6.1.1 Detailed Description

Counter value

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

- rocm_smi.h

## 6.2 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.*

### 6.2.1 Detailed Description

This structure holds error counts.

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

- rocm_smi.h

## 6.3 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.*

### 6.3.1 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

## 6.4 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]

### 6.4.1 Detailed Description

This structure holds information about clock frequencies.

### 6.4.2 Field Documentation

#### 6.4.2.1 uint32_t rsmi_frequencies_t::num_supported

The number of supported frequencies

#### 6.4.2.2 uint32_t rsmi_frequencies_t::current

The current frequency index

#### 6.4.2.3 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

## 6.5 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)*

### 6.5.1 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

## 6.6 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]

### 6.6.1 Detailed Description

RSMI_NUM_VOLTAGE_CURVE_POINTS number of rsmi_od_vddc_point_t's

### 6.6.2 Field Documentation

#### 6.6.2.1 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

## 6.7 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.*

### 6.7.1 Detailed Description

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

### 6.7.2 Field Documentation

#### 6.7.2.1 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

## 6.8 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]

### 6.8.1 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.

### 6.8.2 Field Documentation

#### 6.8.2.1 rsmi_frequencies_t rsmi_pcie_bandwidth_t::transfer_rate

Transfer rates (T/s) that are possible

#### 6.8.2.2 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

## 6.9 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

### 6.9.1 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.

### 6.9.2 Field Documentation

#### 6.9.2.1 rsmi_bit_field_t rsmi_power_profile_status_t::available_profiles

Which profiles are supported by this system

#### 6.9.2.2 rsmi_power_profile_preset_masks_t rsmi_power_profile_status_t::current

Which power profile is currently active

#### 6.9.2.3 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

## 6.10 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.*

### 6.10.1 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

## 6.11 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.*

### 6.11.1 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>
```

### Data Structures

- struct rsmi_counter_value_t
- 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.*

## 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 RSMI_MAX_NUM_POWER_PROFILES (sizeof(rsmi_bit_field_t) * 8)

  *Number of possible power profiles that a system could support.*

## 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.*

## 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_UNKNOWN_ERROR = 0xFFFFFFFF }

  *Error codes retured by rocm_smi_lib functions.*
- enum rsmi_init_flags_t { RSMI_INIT_FLAG_ALL_GPUS = 0x1 }

  *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_PEAK, RSMI_DEV_PERF_LEVEL_STABLE_MIN_MCLK,
  RSMI_DEV_PERF_LEVEL_STABLE_MIN_SCLK, **RSMI_DEV_PERF_LEVEL_LAST** = RSMI_DEV_PERF_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_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, R↩
  SMI_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_TEM↩
  P_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 }

  *This ennumeration is used to indicate from which part of the device a temperature 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, R↩
  SMI_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_PW↩
  R_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_LAST = RSMI_GPU_BLOCK_↩
  GFX, **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_L↩**
  **AST** = RSMI_RAS_ERR_STATE_POISON, **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_T↩
  YPE_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_INV↩
  ALID = 0xFFFFFFFF }

  *This values of this enum are used as frequency identifiers.*

## 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_vendor_name_get (uint32_t id, char *name, size_t len)

  *Get the name string for a give vendor ID.*
- 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_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_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 sensor_ind, 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_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_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_dev_od_freq_range_set (uint32_t dv_ind, rsmi_clk_type_t clk, rsmi_range_t *range)

  *Set the frequency limits 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_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_mask)

  *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.*

### 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 #define RSMI_MAX_FAN_SPEED 255

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

### 7.1.3 Typedef Documentation

#### 7.1.3.1 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 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.

***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_UNKNOWN_ERROR*** An unknown error occurred.

#### 7.1.4.2 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.

#### 7.1.4.3 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

***RSMI_DEV_PERF_LEVEL_MANUAL*** Only use values defined by manually setting the RSMI_CLK_TYP↩ E_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 enum rsmi_sw_component_t

Available clock types.

Software components

**Enumerator**

***RSMI_SW_COMP_DRIVER*** Driver.

#### 7.1.4.5 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 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

***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

#### 7.1.4.7 enum rsmi_counter_command_t

Event counter commands

**Enumerator**

***RSMI_CNTR_CMD_START*** Start the counter.

***RSMI_CNTR_CMD_STOP*** Stop the counter.

#### 7.1.4.8 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.

#### 7.1.4.9 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.

#### 7.1.4.10 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.

#### 7.1.4.11 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.

#### 7.1.4.12 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_LAST*** The highest bit position for supported blocks

#### 7.1.4.13 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.

#### 7.1.4.14 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.

#### 7.1.4.15 enum rsmi_freq_ind_t

This 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.

# Index