

ROCmSMI

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>ROCm System Management Interface (ROCm SMI) Library</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>5</b>
2.1	Modules . . . . .	5
<b>3</b>	<b>Data Structure Index</b>	<b>7</b>
3.1	Data Structures . . . . .	7
<b>4</b>	<b>File Index</b>	<b>9</b>
4.1	File List . . . . .	9
<b>5</b>	<b>Module Documentation</b>	<b>11</b>
5.1	Initialization and Shutdown . . . . .	11
5.1.1	Detailed Description . . . . .	11
5.1.2	Function Documentation . . . . .	11
5.1.2.1	rsmi_init(uint64_t init_flags) . . . . .	11
5.1.2.2	rsmi_shut_down(void) . . . . .	12
5.2	Identifier Queries . . . . .	13
5.2.1	Detailed Description . . . . .	13
5.2.2	Function Documentation . . . . .	13
5.2.2.1	rsmi_num_monitor_devices(uint32_t *num_devices) . . . . .	13
5.2.2.2	rsmi_dev_id_get(uint32_t dv_ind, uint16_t *id) . . . . .	14
5.2.2.3	rsmi_dev_vendor_id_get(uint32_t dv_ind, uint16_t *id) . . . . .	14
5.2.2.4	rsmi_dev_name_get(uint32_t dv_ind, char *name, size_t len) . . . . .	14
5.2.2.5	rsmi_dev_vendor_name_get(uint32_t id, char *name, size_t len) . . . . .	15

5.2.2.6	<code>rsmi_dev_subsystem_id_get(uint32_t dv_ind, uint16_t *id)</code>	15
5.2.2.7	<code>rsmi_dev_subsystem_name_get(uint32_t dv_ind, char *name, size_t len)</code>	16
5.2.2.8	<code>rsmi_dev_subsystem_vendor_id_get(uint32_t dv_ind, uint16_t *id)</code>	16
5.2.2.9	<code>rsmi_dev_unique_id_get(uint32_t dv_ind, uint64_t *id)</code>	16
5.3	PCIe Queries	18
5.3.1	Detailed Description	18
5.3.2	Function Documentation	18
5.3.2.1	<code>rsmi_dev_pci_bandwidth_get(uint32_t dv_ind, rsmi_pcie_bandwidth_t *bandwidth)</code>	18
5.3.2.2	<code>rsmi_dev_pci_id_get(uint32_t dv_ind, uint64_t *bdfid)</code>	18
5.3.2.3	<code>rsmi_dev_pci_throughput_get(uint32_t dv_ind, uint64_t *sent, uint64_t *received, uint64_t *max_pkt_sz)</code>	19
5.3.2.4	<code>rsmi_dev_pci_replay_counter_get(uint32_t dv_ind, uint64_t *counter)</code>	19
5.4	PCIe Control	20
5.4.1	Detailed Description	20
5.4.2	Function Documentation	20
5.4.2.1	<code>rsmi_dev_pci_bandwidth_set(uint32_t dv_ind, uint64_t bw_bitmask)</code>	20
5.5	Power Queries	21
5.5.1	Detailed Description	21
5.5.2	Function Documentation	21
5.5.2.1	<code>rsmi_dev_power_ave_get(uint32_t dv_ind, uint32_t sensor_ind, uint64_t *power)</code>	21
5.5.2.2	<code>rsmi_dev_power_cap_get(uint32_t dv_ind, uint32_t sensor_ind, uint64_t *cap)</code>	21
5.5.2.3	<code>rsmi_dev_power_cap_range_get(uint32_t dv_ind, uint32_t sensor_ind, uint64_t *max, uint64_t *min)</code>	22
5.6	Power Control	23
5.6.1	Detailed Description	23
5.6.2	Function Documentation	23
5.6.2.1	<code>rsmi_dev_power_cap_set(uint32_t dv_ind, uint32_t sensor_ind, uint64_t cap)</code>	23
5.6.2.2	<code>rsmi_dev_power_profile_set(uint32_t dv_ind, uint32_t sensor_ind, rsmi_power_profile_preset_masks_t profile)</code>	23
5.7	Memory Queries	25
5.7.1	Detailed Description	25
5.7.2	Function Documentation	25

5.7.2.1	<code>rsmi_dev_memory_total_get(uint32_t dv_ind, rsmi_memory_type_t mem_type, uint64_t *total)</code>	25
5.7.2.2	<code>rsmi_dev_memory_usage_get(uint32_t dv_ind, rsmi_memory_type_t mem_type, uint64_t *used)</code>	25
5.7.2.3	<code>rsmi_dev_memory_busy_percent_get(uint32_t dv_ind, uint32_t *busy_percent)</code>	26
5.8	Physical State Queries	27
5.8.1	Detailed Description	27
5.8.2	Function Documentation	27
5.8.2.1	<code>rsmi_dev_fan_rpms_get(uint32_t dv_ind, uint32_t sensor_ind, int64_t *speed)</code>	27
5.8.2.2	<code>rsmi_dev_fan_speed_get(uint32_t dv_ind, uint32_t sensor_ind, int64_t *speed)</code>	27
5.8.2.3	<code>rsmi_dev_fan_speed_max_get(uint32_t dv_ind, uint32_t sensor_ind, uint64_t *max_speed)</code>	28
5.8.2.4	<code>rsmi_dev_temp_metric_get(uint32_t dv_ind, uint32_t sensor_type, rsmi_temperature_metric_t metric, int64_t *temperature)</code>	28
5.9	Physical State Control	30
5.9.1	Detailed Description	30
5.9.2	Function Documentation	30
5.9.2.1	<code>rsmi_dev_fan_reset(uint32_t dv_ind, uint32_t sensor_ind)</code>	30
5.9.2.2	<code>rsmi_dev_fan_speed_set(uint32_t dv_ind, uint32_t sensor_ind, uint64_t speed)</code>	30
5.10	Clock, Power and Performance Queries	32
5.10.1	Detailed Description	32
5.10.2	Function Documentation	32
5.10.2.1	<code>rsmi_dev_busy_percent_get(uint32_t dv_ind, uint32_t *busy_percent)</code>	32
5.10.2.2	<code>rsmi_dev_perf_level_get(uint32_t dv_ind, rsmi_dev_perf_level_t *perf)</code>	32
5.10.2.3	<code>rsmi_dev_overdrive_level_get(uint32_t dv_ind, uint32_t *od)</code>	33
5.10.2.4	<code>rsmi_dev_gpu_clk_freq_get(uint32_t dv_ind, rsmi_clk_type_t clk_type, rsmi_frequencies_t *f)</code>	33
5.10.2.5	<code>rsmi_dev_od_volt_info_get(uint32_t dv_ind, rsmi_od_volt_freq_data_t *odv)</code>	34
5.10.2.6	<code>rsmi_dev_od_volt_curve_regions_get(uint32_t dv_ind, uint32_t *num_regions, rsmi_freq_volt_region_t *buffer)</code>	34
5.10.2.7	<code>rsmi_dev_power_profile_presets_get(uint32_t dv_ind, uint32_t sensor_ind, rsmi_power_profile_status_t *status)</code>	34
5.11	Clock, Power and Performance Control	36

5.11.1 Detailed Description . . . . .	36
5.11.2 Function Documentation . . . . .	36
5.11.2.1 rsmi_dev_perf_level_set(int32_t dv_ind, rsmi_dev_perf_level_t perf_lvl) . . . . .	36
5.11.2.2 rsmi_dev_overdrive_level_set(int32_t dv_ind, uint32_t od) . . . . .	36
5.11.2.3 rsmi_dev_gpu_clk_freq_set(uint32_t dv_ind, rsmi_clk_type_t clk_type, uint64_t freq_bitmask) . . . . .	37
5.11.2.4 rsmi_dev_od_freq_range_set(uint32_t dv_ind, rsmi_clk_type_t clk, rsmi_range_t *range) . . . . .	37
5.12 Version Queries . . . . .	39
5.12.1 Detailed Description . . . . .	39
5.12.2 Function Documentation . . . . .	39
5.12.2.1 rsmi_version_get(rsmi_version_t *version) . . . . .	39
5.12.2.2 rsmi_version_str_get(rsmi_sw_component_t component, char *ver_str, uint32_t len) . . . . .	39
5.12.2.3 rsmi_dev_vbios_version_get(uint32_t dv_ind, char *vbios, uint32_t len) . . . . .	40
5.13 Error Queries . . . . .	41
5.13.1 Detailed Description . . . . .	41
5.13.2 Function Documentation . . . . .	41
5.13.2.1 rsmi_dev_ecc_count_get(uint32_t dv_ind, rsmi_gpu_block_t block, rsmi_error_count_t *ec) . . . . .	41
5.13.2.2 rsmi_dev_ecc_enabled_get(uint32_t dv_ind, uint64_t *enabled_mask) . . . . .	41
5.13.2.3 rsmi_dev_ecc_status_get(uint32_t dv_ind, rsmi_gpu_block_t block, rsmi_ras_err_state_t *state) . . . . .	42
5.13.2.4 rsmi_status_string(rsmi_status_t status, const char **status_string) . . . . .	42
5.14 Performance Counter Functions . . . . .	43
5.14.1 Detailed Description . . . . .	43
5.14.2 Function Documentation . . . . .	43
5.14.2.1 rsmi_dev_counter_group_supported(uint32_t dv_ind, rsmi_event_group_t group) . . . . .	43
5.14.2.2 rsmi_dev_counter_create(uint32_t dv_ind, rsmi_event_type_t type, rsmi_event_handle_t *evnt_handle) . . . . .	44
5.14.2.3 rsmi_dev_counter_destroy(rsmi_event_handle_t evnt_handle) . . . . .	44
5.14.2.4 rsmi_counter_control(rsmi_event_handle_t evt_handle, rsmi_counter_command_t cmd, void *cmd_args) . . . . .	44

5.14.2.5	<code>rsmi_counter_read(rsmi_event_handle_t evt_handle, rsmi_counter_value_t *value)</code>	45
5.14.2.6	<code>rsmi_counter_available_counters_get(uint32_t dv_ind, rsmi_event_group_t grp, uint32_t *available)</code>	45
5.15	System Information Functions	46
5.15.1	Detailed Description	46
5.15.2	Function Documentation	46
5.15.2.1	<code>rsmi_dev_compute_process_info_get(rsmi_process_info_t *procs, uint32_t *num_items)</code>	46
5.15.2.2	<code>rsmi_dev_compute_process_info_by_pid_get(uint32_t pid, rsmi_process_info_t *proc)</code>	46
5.16	XGMI Functions	49
5.16.1	Detailed Description	49
5.16.2	Function Documentation	49
5.16.2.1	<code>rsmi_dev_xgmi_error_status(uint32_t dv_ind, rsmi_xgmi_status_t *status)</code>	49
5.16.2.2	<code>rsmi_dev_xgmi_error_reset(uint32_t dv_ind)</code>	49
<b>6</b>	<b>Data Structure Documentation</b>	<b>51</b>
6.1	<code>rsmi_counter_value_t</code> Struct Reference	51
6.1.1	Detailed Description	51
6.2	<code>rsmi_error_count_t</code> Struct Reference	51
6.2.1	Detailed Description	52
6.3	<code>rsmi_freq_volt_region_t</code> Struct Reference	52
6.3.1	Detailed Description	52
6.4	<code>rsmi_frequencies_t</code> Struct Reference	52
6.4.1	Detailed Description	53
6.4.2	Field Documentation	53
6.4.2.1	<code>num_supported</code>	53
6.4.2.2	<code>current</code>	53
6.4.2.3	<code>frequency</code>	53
6.5	<code>rsmi_od_vddc_point_t</code> Struct Reference	53
6.5.1	Detailed Description	53
6.6	<code>rsmi_od_volt_curve_t</code> Struct Reference	54

6.6.1	Detailed Description	54
6.6.2	Field Documentation	54
6.6.2.1	vc_points	54
6.7	rsmi_od_volt_freq_data_t Struct Reference	54
6.7.1	Detailed Description	55
6.7.2	Field Documentation	55
6.7.2.1	curr_mclk_range	55
6.8	rsmi_pcie_bandwidth_t Struct Reference	55
6.8.1	Detailed Description	55
6.8.2	Field Documentation	55
6.8.2.1	transfer_rate	55
6.8.2.2	lanes	55
6.9	rsmi_power_profile_status_t Struct Reference	56
6.9.1	Detailed Description	56
6.9.2	Field Documentation	56
6.9.2.1	available_profiles	56
6.9.2.2	current	56
6.9.2.3	num_profiles	56
6.10	rsmi_process_info_t Struct Reference	56
6.10.1	Detailed Description	57
6.11	rsmi_range_t Struct Reference	57
6.11.1	Detailed Description	57
6.12	rsmi_version_t Struct Reference	57
6.12.1	Detailed Description	58



<b>7 File Documentation</b>	<b>59</b>
7.1 rocm_smi.h File Reference	59
7.1.1 Detailed Description	64
7.1.2 Macro Definition Documentation	64
7.1.2.1 RSMI_MAX_FAN_SPEED	64
7.1.3 Typedef Documentation	65
7.1.3.1 rsmi_event_handle_t	65
7.1.4 Enumeration Type Documentation	65
7.1.4.1 rsmi_status_t	65
7.1.4.2 rsmi_init_flags_t	65
7.1.4.3 rsmi_dev_perf_level_t	66
7.1.4.4 rsmi_sw_component_t	66
7.1.4.5 rsmi_event_group_t	66
7.1.4.6 rsmi_event_type_t	66
7.1.4.7 rsmi_counter_command_t	67
7.1.4.8 rsmi_clk_type_t	67
7.1.4.9 rsmi_temperature_metric_t	67
7.1.4.10 rsmi_temperature_type_t	68
7.1.4.11 rsmi_power_profile_preset_masks_t	68
7.1.4.12 rsmi_gpu_block_t	68
7.1.4.13 rsmi_ras_err_state_t	68
7.1.4.14 rsmi_memory_type_t	69
7.1.4.15 rsmi_freq_ind_t	69
<b>Index</b>	<b>71</b>



## 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:

Initialization and Shutdown . . . . .	11
Identifier Queries . . . . .	13
PCIe Queries . . . . .	18
PCIe Control . . . . .	20
Power Queries . . . . .	21
Power Control . . . . .	23
Memory Queries . . . . .	25
Physical State Queries . . . . .	27
Physical State Control . . . . .	30
Clock, Power and Performance Queries . . . . .	32
Clock, Power and Performance Control . . . . .	36
Version Queries . . . . .	39
Error Queries . . . . .	41
Performance Counter Functions . . . . .	43
System Information Functions . . . . .	46
XGMI Functions . . . . .	49





## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">rsmi_counter_value_t</a> . . . . .	51
<a href="#">rsmi_error_count_t</a>	
This structure holds error counts . . . . .	51
<a href="#">rsmi_freq_volt_region_t</a>	
This structure holds 2 <a href="#">rsmi_range_t</a> 's, one for frequency and one for voltage. These 2 ranges indicate the range of possible values for the corresponding <a href="#">rsmi_od_vddc_point_t</a> . . . . .	52
<a href="#">rsmi_frequencies_t</a>	
This structure holds information about clock frequencies . . . . .	52
<a href="#">rsmi_od_vddc_point_t</a>	
This structure represents a point on the frequency-voltage plane . . . . .	53
<a href="#">rsmi_od_volt_curve_t</a> . . . . .	54
<a href="#">rsmi_od_volt_freq_data_t</a>	
This structure holds the frequency-voltage values for a device . . . . .	54
<a href="#">rsmi_pcie_bandwidth_t</a>	
This structure holds information about the possible PCIe bandwidths. Specifically, the possible transfer rates and their associated numbers of lanes are stored here . . . . .	55
<a href="#">rsmi_power_profile_status_t</a>	
This structure contains information about which power profiles are supported by the system for a given device, and which power profile is currently active . . . . .	56
<a href="#">rsmi_process_info_t</a>	
This structure contains information specific to a process . . . . .	56
<a href="#">rsmi_range_t</a>	
This structure represents a range (e.g., frequencies or voltages) . . . . .	57
<a href="#">rsmi_version_t</a>	
This structure holds version information . . . . .	57



## Chapter 4

# File Index

### 4.1 File List

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

[rocm\\_smi.h](#)

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 . . . . .

59



# 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	<i>init_flags</i>	Bit flags that tell SMI how to initialize. Values of <a href="#">rsmi_init_flags_t</a> may be OR'd together and passed through <i>init_flags</i> to modify how RSMI initializes.
----	-------------------	--

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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 subsystem.*
- [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	<code>num_devices</code>	Caller provided pointer to uint32_t. Upon successful call, the value <code>num_devices</code> will contain the number of monitor devices.
---------	--------------------------	---

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in, out	<i>id</i>	a pointer to <code>uint64_t</code> to which the device id will be written

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<i>dv_ind</i>	a device index
in, out	<i>id</i>	a pointer to <code>uint64_t</code> to which the device vendor id will be written

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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 accomplished with "sudo update-pciids".

#### Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>name</i>	a pointer to a caller provided char buffer to which the name will be written
in	<i>len</i>	the length of the caller provided buffer <code>name</code> .



## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call.
<a href="#"><i>RSMI_STATUS_INSUFFICIENT_SIZE</i></a>	is returned if <code>len</code> bytes is not large enough to hold the entire name. In this case, only <code>len</code> 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 accomplished with "sudo update-pciids".

## Parameters

in	<i>id</i>	a vendor ID
in, out	<i>name</i>	a pointer to a caller provided char buffer to which the name will be written
in	<i>len</i>	the length of the caller provided buffer <code>name</code> .

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call.
<a href="#"><i>RSMI_STATUS_INSUFFICIENT_SIZE</i></a>	is returned if <code>len</code> bytes is not large enough to hold the entire name. In this case, only <code>len</code> 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	<i>dv_ind</i>	a device index
in, out	<i>id</i>	a pointer to <code>uint64_t</code> to which the subsystem device id will be written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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 subsystem.

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 accomplished with "sudo update-pciids".

#### Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>name</i>	a pointer to a caller provided char buffer to which the name will be written
in	<i>len</i>	the length of the caller provided buffer <code>name</code> .

#### Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call.
<a href="#"><i>RSMI_STATUS_INSUFFICIENT_SIZE</i></a>	is returned if <code>len</code> bytes is not large enough to hold the entire name. In this case, only <code>len</code> 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	<i>dv_ind</i>	a device index
in, out	<i>id</i>	a pointer to <code>uint64_t</code> to which the device subsystem vendor id will be written

#### Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in, out	<i>id</i>	a pointer to uint64_t to which the unique ID of the GPU is written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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 [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	<i>dv_ind</i>	a device index
in, out	<i>bandwidth</i>	a pointer to a caller provided <a href="#">rsmi_pcie_bandwidth_t</a> structure to which the frequency information will be written

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in, out	<i>bdfid</i>	a pointer to uint64_t to which the device bdfid value will be written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in, out	<i>sent</i>	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	<i>received</i>	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	<i>max_pkt_sz</i>	a pointer to uint64_t to which the maximum packet size will be written. If pointer is NULL, it will be ignored.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in, out	<i>counter</i>	a pointer to uint64_t to which the sum of the NAK's received and generated by the GPU is written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in	<i>bw_bitmask</i>	A bitmask indicating the indices of the bandwidths that are to be enabled (1) and disabled (0). Only the lowest <a href="#">rsmi_frequencies_t::num_supported</a> (of <a href="#">rsmi_pcie_bandwidth_t</a> ) 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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>power</i>	a pointer to <code>uint64_t</code> to which the average power consumption will be written

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
Generated by Doxygen in, out	<i>cap</i>	a pointer to a <code>uint64_t</code> that indicates the power cap, in microwatts

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>max</i>	a pointer to a <code>uint64_t</code> that indicates the maximum possible power cap, in microwatts
in, out	<i>min</i>	a pointer to a <code>uint64_t</code> that indicates the minimum possible power cap, in microwatts

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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\\_t 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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>cap</i>	a uint64_t that indicates the desired power cap, in microwatts

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>profile</i>	a <a href="#">rsmi_power_profile_preset_masks_t</a> that hold the mask of the desired new power profile

**Return values**

<i>RSMI_STATUS_SUCCESS</i>	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.
- `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.

### 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	<code>dv_ind</code>	a device index
in	<code>mem_type</code>	The type of memory for which the total amount will be found
in, out	<code>total</code>	a pointer to <code>uint64_t</code> to which the total amount of memory will be written

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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 is currently being used to the location pointed to by `total`.

#### Parameters

in	<code>dv_ind</code>	a device index
in	<code>mem_type</code>	The type of memory for which the amount being used will be found
Generated by Doxygen in, out	<code>used</code>	a pointer to <code>uint64_t</code> to which the amount of memory currently being used will be written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call.
--	-----------------------------------

### 5.7.2.3 `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	<i>dv_ind</i>	a device index
in, out	<i>busy_percent</i>	a pointer to the <code>uint32_t</code> to which the busy percent will be written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>speed</i>	a pointer to uint32_t to which the speed will be written

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>speed</i>	a pointer to uint32_t to which the speed will be written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>max_speed</i>	a pointer to uint32_t to which the maximum speed will be written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_type</i>	part of device from which temperature should be obtained. This should come from the enum <a href="#"><code>rsmi_temperature_type_t</code></a>
in	<i>metric</i>	enum indicated which temperature value should be retrieved
in, out	<i>temperature</i>	a pointer to int64_t to which the temperature will be written, in millidegrees Celcius.

Return values

<i>RSMI_STATUS_SUCCESS</i>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>speed</i>	the speed to which the function will attempt to set the fan



Return values

<i>RSMI_STATUS_SUCCESS</i>	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	<code>dv_ind</code>	a device index
in, out	<code>busy_percent</code>	a pointer to the <code>uint32_t</code> to which the busy percent will be written

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in, out	<i>perf</i>	a pointer to <code>rsmi_dev_perf_level_t</code> to which the performance level will be written

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<i>dv_ind</i>	a device index
in, out	<i>od</i>	a pointer to <code>uint32_t</code> to which the overdrive percentage will be written

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<i>dv_ind</i>	a device index
in	<i>clk_type</i>	the type of clock for which the frequency is desired
in, out	<i>f</i>	a pointer to a caller provided <code>rsmi_frequencies_t</code> structure to which the frequency information will be written. Frequency values are in Hz.

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<code>dv_ind</code>	a device index
in	<code>odv</code>	a pointer to an <a href="#">rsmi_od_volt_freq_data_t</a> structure

##### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<code>dv_ind</code>	a device index
in, out	<code>num_regions</code>	As input, this is the number of <a href="#">rsmi_freq_volt_region_t</a> structures that can be written to <code>buffer</code> . As output, this is the number of <a href="#">rsmi_freq_volt_region_t</a> structures that were actually written.
in, out	<code>buffer</code>	a caller provided buffer to which <a href="#">rsmi_freq_volt_region_t</a> structures will be written

##### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in	<i>sensor_ind</i>	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	<i>status</i>	a pointer to <code>rsmi_power_profile_status_t</code> that will be populated by a call to this function

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<code>dv_ind</code>	a device index
in	<code>perf_lvl</code>	the value to which the performance level should be set

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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	<i>dv_ind</i>	a device index
in	<i>od</i>	the value to which the overdrive level should be set

#### Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in	<i>clk_type</i>	the type of clock for which the set of frequencies will be modified
in	<i>freq_bitmask</i>	A bitmask indicating the indices of the frequencies that are to be enabled (1) and disabled (0). Only the lowest <code>rsmi_frequencies_t.num_supported</code> 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	<i>dv_ind</i>	a device index
in	<i>clk</i>	The clock type for which the limits should be imposed.
in	<i>range</i>	A pointer to the <a href="#">rsmi_range_t</a> containing the desired limits

**Return values**

<a href="#">RSMI_STATUS_SUCCESS</a>	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 identifier 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	<code>version</code>	A pointer to an <code>rsmi_version_t</code> structure that will be updated with the version information upon return.
---------	----------------------	--

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<code>component</code>	The component for which the version string is being requested
in, out	<code>ver_str</code>	A pointer to a buffer of char's to which the VBIOS name will be written
in	<code>len</code>	The number of char's pointed to by <code>ver_str</code> which can safely be written to by this function.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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 identifier 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	<i>dv_ind</i>	a device index
in, out	<i>vbios</i>	A pointer to a buffer of char's to which the VBIOS name will be written
in	<i>len</i>	The number of char's pointed to by <code>vbios</code> which can safely be written to by this function.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<code>dv_ind</code>	a device index
in	<code>block</code>	The block for which error counts should be retrieved
in, out	<code>ec</code>	A pointer to an <a href="#">rsmi_error_count_t</a> to which the error counts should be written

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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](#) enumeration 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	<i>dv_ind</i>	a device index
in, out	<i>enabled_mask</i>	A pointer to a uint64_t to which the enabled mask will be written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>dv_ind</i>	a device index
in	<i>block</i>	The block for which error counts should be retrieved
in, out	<i>state</i>	A pointer to an <code>rsmi_ras_err_state_t</code> to which the ECC state should be written

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<i>status</i>	The error status for which a description is desired
in, out	<i>status_string</i>	A pointer to a const char * which will be made to point to a description of the provided error code

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	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	<code>dv_ind</code>	device index of device being queried
in	<code>group</code>	<code>rsmi_event_group_t</code> identifier of group for which support is being queried

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	if the device associatee with <code>dv_ind</code> support counting events of the type indicated by <code>group</code> .
----------------------------------	---

`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	<i>dv_ind</i>	a device index
in	<i>type</i>	the type of performance event to create
in, out	<i>evnt_handle</i>	A pointer to a <code>rsmi_event_handle_t</code> which will be associated with a newly allocated counter

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<i>evnt_handle</i>	handle to event object to be deallocated
----	--------------------	--

#### Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<i>evt_handle</i>	an event handle
in	<i>cmd</i>	The event counter command to be issued
in, out	<i>cmd_args</i>	Currently not used. Should be set to NULL.

## Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<code>evt_handle</code>	an event handle
in, out	<code>value</code>	pointer to memory of size of <code>rsmi_counter_value_t</code> to which the counter value will be written

## Return values

<code>RSMI_STATUS_SUCCESS</code>	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	<code>dv_ind</code>	a device index
in	<code>grp</code>	an event device group
in, out	<code>available</code>	A pointer to a <code>uint32_t</code> to which the number of available counters will be written

## Return values

<code>RSMI_STATUS_SUCCESS</code>	is returned upon successful call
----------------------------------	----------------------------------

## 5.15 System Information Functions

### Functions

- [rsmi\\_status\\_t rsmi\\_dev\\_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\\_dev\\_compute\\_process\\_info\\_by\\_pid\\_get \(uint32\\_t pid, rsmi\\_process\\_info\\_t \\*proc\)](#)  
*Get process information about a specific 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\\_status\\_t rsmi\\_dev\\_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	<code>procs</code>	a pointer to memory provided by the caller to which process information will be written. This may be NULL in which case only <code>num_items</code> will be updated with the number of processes found.
in, out	<code>num_items</code>	A pointer to a <code>uint32_t</code> , which on input, should contain the amount of memory in <a href="#">rsmi_process_info_t</a> 's which have been provided by the <code>procs</code> argument. On output, if <code>procs</code> is non-NULL, this will be updated with the number <a href="#">rsmi_process_info_t</a> structs actually written. If <code>procs</code> is NULL, this argument will be updated with the number processes for which there is information.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	is returned upon successful call
-------------------------------------	----------------------------------

[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\\_status\\_t rsmi\\_dev\\_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	<i>pid</i>	The process ID for which process information is being requested
in, out	<i>proc</i>	a pointer to a <a href="#">rsmi_process_info_t</a> to which process information for <i>pid</i> will be written if it is found.

## Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	is returned upon successful call
-------------------------------------	----------------------------------

[RSMI\\_STATUS\\_NOT\\_FOUND](#) is returned if there was no process information found for the provided *pid*

## 5.16 XGMI Functions

### 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.*

### 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\\_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	<i>dv_ind</i>	a device index
in, out	<i>status</i>	A pointer to an <a href="#">rsmi_xgmi_status_t</a> to which the XGMI error state should be written

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	is returned upon successful call.
-------------------------------------	-----------------------------------

#### 5.16.2.2 [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	<i>dv_ind</i>	a device index
----	---------------	----------------

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	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 the 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\_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.*

### 6.10.1 Detailed Description

This structure contains information specific to a process.

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

- [rocm\\_smi.h](#)

## 6.11 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.*

### 6.11.1 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](#)

## 6.12 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.*

### 6.12.1 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.*
- struct [rsmi\\_process\\_info\\_t](#)  
*This structure contains information specific to a process.*

## 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 returned 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_LE↵`  
`VEL_STABLE_PEAK, 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_NO↵`  
`P_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 enumerating 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 millidegrees Celsius.

- 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 enumeration 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 }

The values of this enum are used as frequency identifiers.

- enum *rsmi\_xgmi\_status\_t* { *RSMI\_XGMI\_STATUS\_NO\_ERRORS* = 0, *RSMI\_XGMI\_STATUS\_ERROR*, *R*↵  
*SMI\_XGMI\_STATUS\_MULTIPLE\_ERRORS* }

XGMI Status.

## 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\\_t](#) 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\\_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](#) (uint32\_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](#) (uint32\_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 identifier 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.*

- `rsmi_status_t rsmi_dev_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_dev_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_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.*

### 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 returned 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 be 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 enumerated 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 enumerating 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 millidegrees Celsius.

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 enumeration 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

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.





# Index

- available\_profiles
  - rsmi\_power\_profile\_status\_t, 56
- Clock, Power and Performance Control, 36
  - rsmi\_dev\_gpu\_clk\_freq\_set, 37
  - rsmi\_dev\_od\_freq\_range\_set, 37
  - rsmi\_dev\_overdrive\_level\_set, 36
  - rsmi\_dev\_perf\_level\_set, 36
- Clock, Power and Performance Queries, 32
  - rsmi\_dev\_busy\_percent\_get, 32
  - rsmi\_dev\_gpu\_clk\_freq\_get, 33
  - rsmi\_dev\_od\_volt\_curve\_regions\_get, 34
  - rsmi\_dev\_od\_volt\_info\_get, 33
  - rsmi\_dev\_overdrive\_level\_get, 33
  - rsmi\_dev\_perf\_level\_get, 32
  - rsmi\_dev\_power\_profile\_presets\_get, 34
- curr\_mclk\_range
  - rsmi\_od\_volt\_freq\_data\_t, 55
- current
  - rsmi\_frequencies\_t, 53
  - rsmi\_power\_profile\_status\_t, 56
- Error Queries, 41
  - rsmi\_dev\_ecc\_count\_get, 41
  - rsmi\_dev\_ecc\_enabled\_get, 41
  - rsmi\_dev\_ecc\_status\_get, 42
  - rsmi\_status\_string, 42
- frequency
  - rsmi\_frequencies\_t, 53
- Identifier Queries, 13
  - rsmi\_dev\_id\_get, 13
  - rsmi\_dev\_name\_get, 14
  - rsmi\_dev\_subsystem\_id\_get, 15
  - rsmi\_dev\_subsystem\_name\_get, 15
  - rsmi\_dev\_subsystem\_vendor\_id\_get, 16
  - rsmi\_dev\_unique\_id\_get, 16
  - rsmi\_dev\_vendor\_id\_get, 14
  - rsmi\_dev\_vendor\_name\_get, 15
  - rsmi\_num\_monitor\_devices, 13
- Initialization and Shutdown, 11
  - rsmi\_init, 11
  - rsmi\_shut\_down, 11
- lanes
  - rsmi\_pcie\_bandwidth\_t, 55
- Memory Queries, 25
  - rsmi\_dev\_memory\_busy\_percent\_get, 26
  - rsmi\_dev\_memory\_total\_get, 25
  - rsmi\_dev\_memory\_usage\_get, 25
- num\_profiles
  - rsmi\_power\_profile\_status\_t, 56
- num\_supported
  - rsmi\_frequencies\_t, 53
- PCIe Control, 20
  - rsmi\_dev\_pci\_bandwidth\_set, 20
- PCIe Queries, 18
  - rsmi\_dev\_pci\_bandwidth\_get, 18
  - rsmi\_dev\_pci\_id\_get, 18
  - rsmi\_dev\_pci\_replay\_counter\_get, 19
  - rsmi\_dev\_pci\_throughput\_get, 19
- Performance Counter Functions, 43
  - rsmi\_counter\_available\_counters\_get, 45
  - rsmi\_counter\_control, 44
  - rsmi\_counter\_read, 45
  - rsmi\_dev\_counter\_create, 43
  - rsmi\_dev\_counter\_destroy, 44
  - rsmi\_dev\_counter\_group\_supported, 43
- Physical State Control, 30
  - rsmi\_dev\_fan\_reset, 30
  - rsmi\_dev\_fan\_speed\_set, 30
- Physical State Queries, 27
  - rsmi\_dev\_fan\_rpms\_get, 27
  - rsmi\_dev\_fan\_speed\_get, 27
  - rsmi\_dev\_fan\_speed\_max\_get, 28
  - rsmi\_dev\_temp\_metric\_get, 28
- Power Control, 23
  - rsmi\_dev\_power\_cap\_set, 23
  - rsmi\_dev\_power\_profile\_set, 23
- Power Queries, 21
  - rsmi\_dev\_power\_ave\_get, 21
  - rsmi\_dev\_power\_cap\_get, 21
  - rsmi\_dev\_power\_cap\_range\_get, 22
- RSMI\_CLK\_TYPE\_DCEF
  - rocm\_smi.h, 67
- RSMI\_CLK\_TYPE\_DF
  - rocm\_smi.h, 67
- RSMI\_CLK\_TYPE\_MEM
  - rocm\_smi.h, 67
- RSMI\_CLK\_TYPE\_SOC
  - rocm\_smi.h, 67
- RSMI\_CLK\_TYPE\_SYS
  - rocm\_smi.h, 67
- RSMI\_CNTR\_CMD\_START
  - rocm\_smi.h, 67
- RSMI\_CNTR\_CMD\_STOP

rocm\_smi.h, [67](#)  
 RSMI\_DEV\_PERF\_LEVEL\_AUTO  
     rocm\_smi.h, [66](#)  
 RSMI\_DEV\_PERF\_LEVEL\_HIGH  
     rocm\_smi.h, [66](#)  
 RSMI\_DEV\_PERF\_LEVEL\_LOW  
     rocm\_smi.h, [66](#)  
 RSMI\_DEV\_PERF\_LEVEL\_MANUAL  
     rocm\_smi.h, [66](#)  
 RSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_MCLK  
     rocm\_smi.h, [66](#)  
 RSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_SCLK  
     rocm\_smi.h, [66](#)  
 RSMI\_DEV\_PERF\_LEVEL\_STABLE\_PEAK  
     rocm\_smi.h, [66](#)  
 RSMI\_DEV\_PERF\_LEVEL\_STABLE\_STD  
     rocm\_smi.h, [66](#)  
 RSMI\_DEV\_PERF\_LEVEL\_UNKNOWN  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_GRP\_XGMI  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_XGMI\_0\_BEATS\_TX  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_XGMI\_0\_NOP\_TX  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_XGMI\_0\_REQUEST\_TX  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_XGMI\_0\_RESPONSE\_TX  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_XGMI\_1\_BEATS\_TX  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_XGMI\_1\_NOP\_TX  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_XGMI\_1\_REQUEST\_TX  
     rocm\_smi.h, [66](#)  
 RSMI\_EVNT\_XGMI\_1\_RESPONSE\_TX  
     rocm\_smi.h, [66](#)  
 RSMI\_FREQ\_IND\_INVALID  
     rocm\_smi.h, [69](#)  
 RSMI\_FREQ\_IND\_MAX  
     rocm\_smi.h, [69](#)  
 RSMI\_FREQ\_IND\_MIN  
     rocm\_smi.h, [69](#)  
 RSMI\_GPU\_BLOCK\_GFX  
     rocm\_smi.h, [68](#)  
 RSMI\_GPU\_BLOCK\_INVALID  
     rocm\_smi.h, [68](#)  
 RSMI\_GPU\_BLOCK\_LAST  
     rocm\_smi.h, [68](#)  
 RSMI\_GPU\_BLOCK\_SDMA  
     rocm\_smi.h, [68](#)  
 RSMI\_GPU\_BLOCK\_UMC  
     rocm\_smi.h, [68](#)  
 RSMI\_INIT\_FLAG\_ALL\_GPUS  
     rocm\_smi.h, [65](#)  
 RSMI\_MAX\_FAN\_SPEED  
     rocm\_smi.h, [64](#)  
 RSMI\_MEM\_TYPE\_GTT  
     rocm\_smi.h, [69](#)  
 RSMI\_MEM\_TYPE\_VIS\_VRAM  
     rocm\_smi.h, [69](#)  
 RSMI\_MEM\_TYPE\_VRAM  
     rocm\_smi.h, [69](#)  
 RSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT  
     rocm\_smi.h, [68](#)  
 RSMI\_PWR\_PROF\_PRST\_COMPUTE\_MASK  
     rocm\_smi.h, [68](#)  
 RSMI\_PWR\_PROF\_PRST\_CUSTOM\_MASK  
     rocm\_smi.h, [68](#)  
 RSMI\_PWR\_PROF\_PRST\_LAST  
     rocm\_smi.h, [68](#)  
 RSMI\_PWR\_PROF\_PRST\_POWER\_SAVING\_MASK  
     rocm\_smi.h, [68](#)  
 RSMI\_PWR\_PROF\_PRST\_VIDEO\_MASK  
     rocm\_smi.h, [68](#)  
 RSMI\_PWR\_PROF\_PRST\_VR\_MASK  
     rocm\_smi.h, [68](#)  
 RSMI\_RAS\_ERR\_STATE\_DISABLED  
     rocm\_smi.h, [68](#)  
 RSMI\_RAS\_ERR\_STATE\_MULT\_UC  
     rocm\_smi.h, [68](#)  
 RSMI\_RAS\_ERR\_STATE\_NONE  
     rocm\_smi.h, [68](#)  
 RSMI\_RAS\_ERR\_STATE\_PARITY  
     rocm\_smi.h, [68](#)  
 RSMI\_RAS\_ERR\_STATE\_POISON  
     rocm\_smi.h, [68](#)  
 RSMI\_RAS\_ERR\_STATE\_SING\_C  
     rocm\_smi.h, [68](#)  
 RSMI\_STATUS\_FILE\_ERROR  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_INIT\_ERROR  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_INPUT\_OUT\_OF\_BOUNDS  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_INSUFFICIENT\_SIZE  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_INTERNAL\_EXCEPTION  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_INTERRUPT  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_INVALID\_ARGS  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_NOT\_FOUND  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_NOT\_SUPPORTED  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_NOT\_YET\_IMPLEMENTED  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_OUT\_OF\_RESOURCES  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_PERMISSION  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_SUCCESS  
     rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_UNEXPECTED\_SIZE

rocm\_smi.h, [65](#)  
 RSMI\_STATUS\_UNKNOWN\_ERROR  
     rocm\_smi.h, [65](#)  
 RSMI\_SW\_COMP\_DRIVER  
     rocm\_smi.h, [66](#)  
 RSMI\_TEMP\_CRIT\_MIN\_HYST  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_CRIT\_MIN  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_CRITICAL\_HYST  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_CRITICAL  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_CURRENT  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_EMERGENCY\_HYST  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_EMERGENCY  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_HIGHEST  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_LOWEST  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_MAX\_HYST  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_MAX  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_MIN\_HYST  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_MIN  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_OFFSET  
     rocm\_smi.h, [67](#)  
 RSMI\_TEMP\_TYPE\_EDGE  
     rocm\_smi.h, [68](#)  
 RSMI\_TEMP\_TYPE\_JUNCTION  
     rocm\_smi.h, [68](#)  
 RSMI\_TEMP\_TYPE\_MEMORY  
     rocm\_smi.h, [68](#)  
 rocm\_smi.h, [59](#)  
     RSMI\_CLK\_TYPE\_DCEF, [67](#)  
     RSMI\_CLK\_TYPE\_DF, [67](#)  
     RSMI\_CLK\_TYPE\_MEM, [67](#)  
     RSMI\_CLK\_TYPE\_SOC, [67](#)  
     RSMI\_CLK\_TYPE\_SYS, [67](#)  
     RSMI\_CNTR\_CMD\_START, [67](#)  
     RSMI\_CNTR\_CMD\_STOP, [67](#)  
     RSMI\_DEV\_PERF\_LEVEL\_AUTO, [66](#)  
     RSMI\_DEV\_PERF\_LEVEL\_HIGH, [66](#)  
     RSMI\_DEV\_PERF\_LEVEL\_LOW, [66](#)  
     RSMI\_DEV\_PERF\_LEVEL\_MANUAL, [66](#)  
     RSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_MCLK,  
         [66](#)  
     RSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_SCLK,  
         [66](#)  
     RSMI\_DEV\_PERF\_LEVEL\_STABLE\_PEAK, [66](#)  
     RSMI\_DEV\_PERF\_LEVEL\_STABLE\_STD, [66](#)  
     RSMI\_DEV\_PERF\_LEVEL\_UNKNOWN, [66](#)  
     RSMI\_EVNT\_GRP\_XGMI, [66](#)  
     RSMI\_EVNT\_XGMI\_0\_BEATS\_TX, [66](#)  
     RSMI\_EVNT\_XGMI\_0\_NOP\_TX, [66](#)  
     RSMI\_EVNT\_XGMI\_0\_REQUEST\_TX, [66](#)  
     RSMI\_EVNT\_XGMI\_0\_RESPONSE\_TX, [66](#)  
     RSMI\_EVNT\_XGMI\_1\_BEATS\_TX, [66](#)  
     RSMI\_EVNT\_XGMI\_1\_NOP\_TX, [66](#)  
     RSMI\_EVNT\_XGMI\_1\_REQUEST\_TX, [66](#)  
     RSMI\_EVNT\_XGMI\_1\_RESPONSE\_TX, [66](#)  
     RSMI\_FREQ\_IND\_INVALID, [69](#)  
     RSMI\_FREQ\_IND\_MAX, [69](#)  
     RSMI\_FREQ\_IND\_MIN, [69](#)  
     RSMI\_GPU\_BLOCK\_GFX, [68](#)  
     RSMI\_GPU\_BLOCK\_INVALID, [68](#)  
     RSMI\_GPU\_BLOCK\_LAST, [68](#)  
     RSMI\_GPU\_BLOCK\_SDMA, [68](#)  
     RSMI\_GPU\_BLOCK\_UMC, [68](#)  
     RSMI\_INIT\_FLAG\_ALL\_GPUS, [65](#)  
     RSMI\_MAX\_FAN\_SPEED, [64](#)  
     RSMI\_MEM\_TYPE\_GTT, [69](#)  
     RSMI\_MEM\_TYPE\_VIS\_VRAM, [69](#)  
     RSMI\_MEM\_TYPE\_VRAM, [69](#)  
     RSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT,  
         [68](#)  
     RSMI\_PWR\_PROF\_PRST\_COMPUTE\_MASK, [68](#)  
     RSMI\_PWR\_PROF\_PRST\_CUSTOM\_MASK, [68](#)  
     RSMI\_PWR\_PROF\_PRST\_LAST, [68](#)  
     RSMI\_PWR\_PROF\_PRST\_POWER\_SAVING\_↵  
         MASK, [68](#)  
     RSMI\_PWR\_PROF\_PRST\_VIDEO\_MASK, [68](#)  
     RSMI\_PWR\_PROF\_PRST\_VR\_MASK, [68](#)  
     RSMI\_RAS\_ERR\_STATE\_DISABLED, [68](#)  
     RSMI\_RAS\_ERR\_STATE\_MULT\_UC, [68](#)  
     RSMI\_RAS\_ERR\_STATE\_NONE, [68](#)  
     RSMI\_RAS\_ERR\_STATE\_PARITY, [68](#)  
     RSMI\_RAS\_ERR\_STATE\_POISON, [68](#)  
     RSMI\_RAS\_ERR\_STATE\_SING\_C, [68](#)  
     RSMI\_STATUS\_FILE\_ERROR, [65](#)  
     RSMI\_STATUS\_INIT\_ERROR, [65](#)  
     RSMI\_STATUS\_INPUT\_OUT\_OF\_BOUNDS, [65](#)  
     RSMI\_STATUS\_INSUFFICIENT\_SIZE, [65](#)  
     RSMI\_STATUS\_INTERNAL\_EXCEPTION, [65](#)  
     RSMI\_STATUS\_INTERRUPT, [65](#)  
     RSMI\_STATUS\_INVALID\_ARGS, [65](#)  
     RSMI\_STATUS\_NOT\_FOUND, [65](#)  
     RSMI\_STATUS\_NOT\_SUPPORTED, [65](#)  
     RSMI\_STATUS\_NOT\_YET\_IMPLEMENTED, [65](#)  
     RSMI\_STATUS\_OUT\_OF\_RESOURCES, [65](#)  
     RSMI\_STATUS\_PERMISSION, [65](#)  
     RSMI\_STATUS\_SUCCESS, [65](#)  
     RSMI\_STATUS\_UNEXPECTED\_SIZE, [65](#)  
     RSMI\_STATUS\_UNKNOWN\_ERROR, [65](#)  
     RSMI\_SW\_COMP\_DRIVER, [66](#)  
     RSMI\_TEMP\_CRIT\_MIN\_HYST, [67](#)  
     RSMI\_TEMP\_CRIT\_MIN, [67](#)  
     RSMI\_TEMP\_CRITICAL\_HYST, [67](#)  
     RSMI\_TEMP\_CRITICAL, [67](#)  
     RSMI\_TEMP\_CURRENT, [67](#)

- RSMI\_TEMP\_EMERGENCY\_HYST, 67
- RSMI\_TEMP\_EMERGENCY, 67
- RSMI\_TEMP\_HIGHEST, 67
- RSMI\_TEMP\_LOWEST, 67
- RSMI\_TEMP\_MAX\_HYST, 67
- RSMI\_TEMP\_MAX, 67
- RSMI\_TEMP\_MIN\_HYST, 67
- RSMI\_TEMP\_MIN, 67
- RSMI\_TEMP\_OFFSET, 67
- RSMI\_TEMP\_TYPE\_EDGE, 68
- RSMI\_TEMP\_TYPE\_JUNCTION, 68
- RSMI\_TEMP\_TYPE\_MEMORY, 68
- rsmi\_clk\_type\_t, 67
- rsmi\_counter\_command\_t, 66
- rsmi\_dev\_perf\_level\_t, 65
- rsmi\_event\_group\_t, 66
- rsmi\_event\_handle\_t, 65
- rsmi\_event\_type\_t, 66
- rsmi\_freq\_ind\_t, 69
- rsmi\_gpu\_block\_t, 68
- rsmi\_init\_flags\_t, 65
- rsmi\_memory\_type\_t, 68
- rsmi\_power\_profile\_preset\_masks\_t, 68
- rsmi\_ras\_err\_state\_t, 68
- rsmi\_status\_t, 65
- rsmi\_sw\_component\_t, 66
- rsmi\_temperature\_metric\_t, 67
- rsmi\_temperature\_type\_t, 67
- rsmi\_clk\_type\_t
  - rocm\_smi.h, 67
- rsmi\_counter\_available\_counters\_get
  - Performance Counter Functions, 45
- rsmi\_counter\_command\_t
  - rocm\_smi.h, 66
- rsmi\_counter\_control
  - Performance Counter Functions, 44
- rsmi\_counter\_read
  - Performance Counter Functions, 45
- rsmi\_counter\_value\_t, 51
- rsmi\_dev\_busy\_percent\_get
  - Clock, Power and Performance Queries, 32
- rsmi\_dev\_compute\_process\_info\_by\_pid\_get
  - System Information Functions, 46
- rsmi\_dev\_compute\_process\_info\_get
  - System Information Functions, 46
- rsmi\_dev\_counter\_create
  - Performance Counter Functions, 43
- rsmi\_dev\_counter\_destroy
  - Performance Counter Functions, 44
- rsmi\_dev\_counter\_group\_supported
  - Performance Counter Functions, 43
- rsmi\_dev\_ecc\_count\_get
  - Error Queries, 41
- rsmi\_dev\_ecc\_enabled\_get
  - Error Queries, 41
- rsmi\_dev\_ecc\_status\_get
  - Error Queries, 42
- rsmi\_dev\_fan\_reset
  - Physical State Control, 30
- rsmi\_dev\_fan\_rpms\_get
  - Physical State Queries, 27
- rsmi\_dev\_fan\_speed\_get
  - Physical State Queries, 27
- rsmi\_dev\_fan\_speed\_max\_get
  - Physical State Queries, 28
- rsmi\_dev\_fan\_speed\_set
  - Physical State Control, 30
- rsmi\_dev\_gpu\_clk\_freq\_get
  - Clock, Power and Performance Queries, 33
- rsmi\_dev\_gpu\_clk\_freq\_set
  - Clock, Power and Performance Control, 37
- rsmi\_dev\_id\_get
  - Identifier Queries, 13
- rsmi\_dev\_memory\_busy\_percent\_get
  - Memory Queries, 26
- rsmi\_dev\_memory\_total\_get
  - Memory Queries, 25
- rsmi\_dev\_memory\_usage\_get
  - Memory Queries, 25
- rsmi\_dev\_name\_get
  - Identifier Queries, 14
- rsmi\_dev\_od\_freq\_range\_set
  - Clock, Power and Performance Control, 37
- rsmi\_dev\_od\_volt\_curve\_regions\_get
  - Clock, Power and Performance Queries, 34
- rsmi\_dev\_od\_volt\_info\_get
  - Clock, Power and Performance Queries, 33
- rsmi\_dev\_overdrive\_level\_get
  - Clock, Power and Performance Queries, 33
- rsmi\_dev\_overdrive\_level\_set
  - Clock, Power and Performance Control, 36
- rsmi\_dev\_pci\_bandwidth\_get
  - PCIe Queries, 18
- rsmi\_dev\_pci\_bandwidth\_set
  - PCIe Control, 20
- rsmi\_dev\_pci\_id\_get
  - PCIe Queries, 18
- rsmi\_dev\_pci\_replay\_counter\_get
  - PCIe Queries, 19
- rsmi\_dev\_pci\_throughput\_get
  - PCIe Queries, 19
- rsmi\_dev\_perf\_level\_get
  - Clock, Power and Performance Queries, 32
- rsmi\_dev\_perf\_level\_set
  - Clock, Power and Performance Control, 36
- rsmi\_dev\_perf\_level\_t
  - rocm\_smi.h, 65
- rsmi\_dev\_power\_ave\_get
  - Power Queries, 21
- rsmi\_dev\_power\_cap\_get
  - Power Queries, 21
- rsmi\_dev\_power\_cap\_range\_get
  - Power Queries, 22
- rsmi\_dev\_power\_cap\_set
  - Power Control, 23
- rsmi\_dev\_power\_profile\_presets\_get

- Clock, Power and Performance Queries, [34](#)
- `rsmi_dev_power_profile_set`
  - Power Control, [23](#)
- `rsmi_dev_subsystem_id_get`
  - Identifier Queries, [15](#)
- `rsmi_dev_subsystem_name_get`
  - Identifier Queries, [15](#)
- `rsmi_dev_subsystem_vendor_id_get`
  - Identifier Queries, [16](#)
- `rsmi_dev_temp_metric_get`
  - Physical State Queries, [28](#)
- `rsmi_dev_unique_id_get`
  - Identifier Queries, [16](#)
- `rsmi_dev_vbios_version_get`
  - Version Queries, [40](#)
- `rsmi_dev_vendor_id_get`
  - Identifier Queries, [14](#)
- `rsmi_dev_vendor_name_get`
  - Identifier Queries, [15](#)
- `rsmi_dev_xgmi_error_reset`
  - XGMI Functions, [49](#)
- `rsmi_dev_xgmi_error_status`
  - XGMI Functions, [49](#)
- `rsmi_error_count_t`, [51](#)
- `rsmi_event_group_t`
  - `rocm_smi.h`, [66](#)
- `rsmi_event_handle_t`
  - `rocm_smi.h`, [65](#)
- `rsmi_event_type_t`
  - `rocm_smi.h`, [66](#)
- `rsmi_freq_ind_t`
  - `rocm_smi.h`, [69](#)
- `rsmi_freq_volt_region_t`, [52](#)
- `rsmi_frequencies_t`, [52](#)
  - current, [53](#)
  - frequency, [53](#)
  - num\_supported, [53](#)
- `rsmi_gpu_block_t`
  - `rocm_smi.h`, [68](#)
- `rsmi_init`
  - Initialization and Shutdown, [11](#)
- `rsmi_init_flags_t`
  - `rocm_smi.h`, [65](#)
- `rsmi_memory_type_t`
  - `rocm_smi.h`, [68](#)
- `rsmi_num_monitor_devices`
  - Identifier Queries, [13](#)
- `rsmi_od_vddc_point_t`, [53](#)
- `rsmi_od_volt_curve_t`, [54](#)
  - vc\_points, [54](#)
- `rsmi_od_volt_freq_data_t`, [54](#)
  - curr\_mclk\_range, [55](#)
- `rsmi_pcie_bandwidth_t`, [55](#)
  - lanes, [55](#)
  - transfer\_rate, [55](#)
- `rsmi_power_profile_preset_masks_t`
  - `rocm_smi.h`, [68](#)
- `rsmi_power_profile_status_t`, [56](#)
  - available\_profiles, [56](#)
  - current, [56](#)
  - num\_profiles, [56](#)
- `rsmi_process_info_t`, [56](#)
- `rsmi_range_t`, [57](#)
- `rsmi_ras_err_state_t`
  - `rocm_smi.h`, [68](#)
- `rsmi_shut_down`
  - Initialization and Shutdown, [11](#)
- `rsmi_status_string`
  - Error Queries, [42](#)
- `rsmi_status_t`
  - `rocm_smi.h`, [65](#)
- `rsmi_sw_component_t`
  - `rocm_smi.h`, [66](#)
- `rsmi_temperature_metric_t`
  - `rocm_smi.h`, [67](#)
- `rsmi_temperature_type_t`
  - `rocm_smi.h`, [67](#)
- `rsmi_version_get`
  - Version Queries, [39](#)
- `rsmi_version_str_get`
  - Version Queries, [39](#)
- `rsmi_version_t`, [57](#)
- System Information Functions, [46](#)
  - `rsmi_dev_compute_process_info_by_pid_get`, [46](#)
  - `rsmi_dev_compute_process_info_get`, [46](#)
- transfer\_rate
  - `rsmi_pcie_bandwidth_t`, [55](#)
- vc\_points
  - `rsmi_od_volt_curve_t`, [54](#)
- Version Queries, [39](#)
  - `rsmi_dev_vbios_version_get`, [40](#)
  - `rsmi_version_get`, [39](#)
  - `rsmi_version_str_get`, [39](#)
- XGMI Functions, [49](#)
  - `rsmi_dev_xgmi_error_reset`, [49](#)
  - `rsmi_dev_xgmi_error_status`, [49](#)