

ROCmSMI

Generated by Doxygen 1.9.7



<b>1 ROCm System Management Interface (ROCm SMI) Library</b>	<b>1</b>
1.1 DISCLAIMER	1
1.2 Building ROCm SMI	1
1.3 Usage Basics	3
1.3.1 Device Indices	3
1.4 Hello ROCm SMI	3
<b>2 Deprecated List</b>	<b>5</b>
<b>3 Module Index</b>	<b>7</b>
3.1 Modules	7
<b>4 Data Structure Index</b>	<b>9</b>
4.1 Data Structures	9
<b>5 File Index</b>	<b>11</b>
5.1 File List	11
<b>6 Module Documentation</b>	<b>13</b>
6.1 Initialization and Shutdown	13
6.1.1 Detailed Description	13
6.1.2 Function Documentation	13
6.1.2.1 rsmi_init()	13
6.1.2.2 rsmi_shut_down()	14
6.2 Identifier Queries	14
6.2.1 Detailed Description	14
6.2.2 Function Documentation	15
6.2.2.1 rsmi_num_monitor_devices()	15
6.2.2.2 rsmi_dev_id_get()	15
6.2.2.3 rsmi_dev_sku_get()	16
6.2.2.4 rsmi_dev_vendor_id_get()	16
6.2.2.5 rsmi_dev_name_get()	17
6.2.2.6 rsmi_dev_brand_get()	17
6.2.2.7 rsmi_dev_vendor_name_get()	18
6.2.2.8 rsmi_dev_vram_vendor_get()	19
6.2.2.9 rsmi_dev_serial_number_get()	19
6.2.2.10 rsmi_dev_subsystem_id_get()	20
6.2.2.11 rsmi_dev_subsystem_name_get()	20
6.2.2.12 rsmi_dev_drm_render_minor_get()	21
6.2.2.13 rsmi_dev_subsystem_vendor_id_get()	21
6.2.2.14 rsmi_dev_unique_id_get()	22
6.3 PCIe Queries	22
6.3.1 Detailed Description	23
6.3.2 Function Documentation	23

6.3.2.1 rsmi_dev_pci_bandwidth_get()	23
6.3.2.2 rsmi_dev_pci_id_get()	23
6.3.2.3 rsmi_topo_numa_affinity_get()	24
6.3.2.4 rsmi_dev_pci_throughput_get()	25
6.3.2.5 rsmi_dev_pci_replay_counter_get()	25
6.4 PCIe Control	26
6.4.1 Detailed Description	26
6.4.2 Function Documentation	26
6.4.2.1 rsmi_dev_pci_bandwidth_set()	26
6.5 Power Queries	27
6.5.1 Detailed Description	27
6.5.2 Function Documentation	27
6.5.2.1 rsmi_dev_power_ave_get()	27
6.5.2.2 rsmi_dev_energy_count_get()	28
6.5.2.3 rsmi_dev_power_cap_get()	28
6.5.2.4 rsmi_dev_power_cap_default_get()	29
6.5.2.5 rsmi_dev_power_cap_range_get()	29
6.6 Power Control	30
6.6.1 Detailed Description	30
6.6.2 Function Documentation	31
6.6.2.1 rsmi_dev_power_cap_set()	31
6.6.2.2 rsmi_dev_power_profile_set()	31
6.7 Memory Queries	32
6.7.1 Detailed Description	32
6.7.2 Function Documentation	32
6.7.2.1 rsmi_dev_memory_total_get()	32
6.7.2.2 rsmi_dev_memory_usage_get()	33
6.7.2.3 rsmi_dev_memory_busy_percent_get()	33
6.7.2.4 rsmi_dev_memory_reserved_pages_get()	34
6.8 Physical State Queries	35
6.8.1 Detailed Description	35
6.8.2 Function Documentation	35
6.8.2.1 rsmi_dev_fan_rpms_get()	35
6.8.2.2 rsmi_dev_fan_speed_get()	36
6.8.2.3 rsmi_dev_fan_speed_max_get()	36
6.8.2.4 rsmi_dev_temp_metric_get()	37
6.8.2.5 rsmi_dev_volt_metric_get()	38
6.9 Physical State Control	38
6.9.1 Detailed Description	38
6.9.2 Function Documentation	39
6.9.2.1 rsmi_dev_fan_reset()	39
6.9.2.2 rsmi_dev_fan_speed_set()	39

6.10 Clock, Power and Performance Queries . . . . .	40
6.10.1 Detailed Description . . . . .	40
6.10.2 Function Documentation . . . . .	41
6.10.2.1 rsmi_dev_busy_percent_get() . . . . .	41
6.10.2.2 rsmi_utilization_count_get() . . . . .	41
6.10.2.3 rsmi_dev_perf_level_get() . . . . .	42
6.10.2.4 rsmi_perf_determinism_mode_set() . . . . .	42
6.10.2.5 rsmi_dev_overdrive_level_get() . . . . .	43
6.10.2.6 rsmi_dev_mem_overdrive_level_get() . . . . .	43
6.10.2.7 rsmi_dev_gpu_clk_freq_get() . . . . .	44
6.10.2.8 rsmi_dev_gpu_reset() . . . . .	45
6.10.2.9 rsmi_dev_od_volt_info_get() . . . . .	45
6.10.2.10 rsmi_dev_gpu_metrics_info_get() . . . . .	45
6.10.2.11 rsmi_dev_clk_range_set() . . . . .	46
6.10.2.12 rsmi_dev_od_clk_info_set() . . . . .	47
6.10.2.13 rsmi_dev_od_volt_info_set() . . . . .	47
6.10.2.14 rsmi_dev_od_volt_curve_regions_get() . . . . .	48
6.10.2.15 rsmi_dev_power_profile_presets_get() . . . . .	48
6.11 Clock, Power and Performance Control . . . . .	49
6.11.1 Detailed Description . . . . .	49
6.11.2 Function Documentation . . . . .	50
6.11.2.1 rsmi_dev_perf_level_set() . . . . .	50
6.11.2.2 rsmi_dev_perf_level_set_v1() . . . . .	50
6.11.2.3 rsmi_dev_overdrive_level_set() . . . . .	51
6.11.2.4 rsmi_dev_overdrive_level_set_v1() . . . . .	52
6.11.2.5 rsmi_dev_gpu_clk_freq_set() . . . . .	52
6.12 Version Queries . . . . .	53
6.12.1 Detailed Description . . . . .	53
6.12.2 Function Documentation . . . . .	53
6.12.2.1 rsmi_version_get() . . . . .	53
6.12.2.2 rsmi_version_str_get() . . . . .	54
6.12.2.3 rsmi_dev_vbios_version_get() . . . . .	54
6.12.2.4 rsmi_dev_firmware_version_get() . . . . .	55
6.13 Error Queries . . . . .	56
6.13.1 Detailed Description . . . . .	56
6.13.2 Function Documentation . . . . .	56
6.13.2.1 rsmi_dev_ecc_count_get() . . . . .	56
6.13.2.2 rsmi_dev_ecc_enabled_get() . . . . .	57
6.13.2.3 rsmi_dev_ecc_status_get() . . . . .	57
6.13.2.4 rsmi_status_string() . . . . .	58
6.14 Performance Counter Functions . . . . .	58
6.14.1 Detailed Description . . . . .	59

6.14.2 Important Notes about Counter Values . . . . .	59
6.14.3 Function Documentation . . . . .	60
6.14.3.1 rsmi_dev_counter_group_supported() . . . . .	60
6.14.3.2 rsmi_dev_counter_create() . . . . .	61
6.14.3.3 rsmi_dev_counter_destroy() . . . . .	61
6.14.3.4 rsmi_counter_control() . . . . .	62
6.14.3.5 rsmi_counter_read() . . . . .	62
6.14.3.6 rsmi_counter_available_counters_get() . . . . .	62
6.15 System Information Functions . . . . .	63
6.15.1 Detailed Description . . . . .	63
6.15.2 Function Documentation . . . . .	63
6.15.2.1 rsmi_compute_process_info_get() . . . . .	63
6.15.2.2 rsmi_compute_process_info_by_pid_get() . . . . .	64
6.15.2.3 rsmi_compute_process_gpus_get() . . . . .	64
6.16 XGMI Functions . . . . .	65
6.16.1 Detailed Description . . . . .	65
6.16.2 Function Documentation . . . . .	66
6.16.2.1 rsmi_dev_xgmi_error_status() . . . . .	66
6.16.2.2 rsmi_dev_xgmi_error_reset() . . . . .	66
6.16.2.3 rsmi_dev_xgmi_hive_id_get() . . . . .	66
6.17 Hardware Topology Functions . . . . .	67
6.17.1 Detailed Description . . . . .	67
6.17.2 Function Documentation . . . . .	68
6.17.2.1 rsmi_topo_get_numa_node_number() . . . . .	68
6.17.2.2 rsmi_topo_get_link_weight() . . . . .	69
6.17.2.3 rsmi_minmax_bandwidth_get() . . . . .	69
6.17.2.4 rsmi_topo_get_link_type() . . . . .	70
6.17.2.5 rsmi_is_P2P_accessible() . . . . .	70
6.18 Compute Partition Functions . . . . .	71
6.18.1 Detailed Description . . . . .	71
6.18.2 Function Documentation . . . . .	71
6.18.2.1 rsmi_dev_compute_partition_get() . . . . .	71
6.18.2.2 rsmi_dev_compute_partition_set() . . . . .	72
6.18.2.3 rsmi_dev_compute_partition_reset() . . . . .	72
6.19 NPS Mode Functions . . . . .	74
6.19.1 Detailed Description . . . . .	74
6.19.2 Function Documentation . . . . .	74
6.19.2.1 rsmi_dev_nps_mode_get() . . . . .	74
6.19.2.2 rsmi_dev_nps_mode_set() . . . . .	75
6.19.2.3 rsmi_dev_nps_mode_reset() . . . . .	75
6.20 Supported Functions . . . . .	76
6.20.1 Detailed Description . . . . .	76

6.20.2 Function Documentation	77
6.20.2.1 rsmi_dev_supported_func_iterator_open()	77
6.20.2.2 rsmi_dev_supported_variant_iterator_open()	79
6.20.2.3 rsmi_func_iter_next()	79
6.20.2.4 rsmi_dev_supported_func_iterator_close()	80
6.20.2.5 rsmi_func_iter_value_get()	80
6.21 Event Notification Functions	81
6.21.1 Detailed Description	81
6.21.2 Function Documentation	81
6.21.2.1 rsmi_event_notification_init()	81
6.21.2.2 rsmi_event_notification_mask_set()	81
6.21.2.3 rsmi_event_notification_get()	82
6.21.2.4 rsmi_event_notification_stop()	83
<b>7 Data Structure Documentation</b>	<b>85</b>
7.1 id Union Reference	85
7.1.1 Detailed Description	85
7.1.2 Field Documentation	86
7.1.2.1 memory_type	86
7.2 metrics_table_header_t Struct Reference	86
7.2.1 Detailed Description	86
7.3 rsmi_counter_value_t Struct Reference	86
7.3.1 Detailed Description	86
7.3.2 Field Documentation	87
7.3.2.1 time_enabled	87
7.3.2.2 time_running	87
7.4 rsmi_error_count_t Struct Reference	87
7.4.1 Detailed Description	87
7.5 rsmi_evt_notification_data_t Struct Reference	87
7.5.1 Detailed Description	88
7.6 rsmi_freq_volt_region_t Struct Reference	88
7.6.1 Detailed Description	88
7.7 rsmi_frequencies_t Struct Reference	88
7.7.1 Detailed Description	89
7.7.2 Field Documentation	89
7.7.2.1 num_supported	89
7.7.2.2 current	89
7.7.2.3 frequency	89
7.8 rsmi_gpu_metrics_t Struct Reference	89
7.9 rsmi_od_vddc_point_t Struct Reference	89
7.9.1 Detailed Description	90
7.10 rsmi_od_volt_curve_t Struct Reference	90

7.10.1 Detailed Description	90
7.10.2 Field Documentation	90
7.10.2.1 vc_points	90
7.11 rsmi_od_volt_freq_data_t Struct Reference	90
7.11.1 Detailed Description	91
7.11.2 Field Documentation	91
7.11.2.1 curr_mclk_range	91
7.12 rsmi_pcie_bandwidth_t Struct Reference	91
7.12.1 Detailed Description	91
7.12.2 Field Documentation	92
7.12.2.1 transfer_rate	92
7.12.2.2 lanes	92
7.13 rsmi_power_profile_status_t Struct Reference	92
7.13.1 Detailed Description	92
7.13.2 Field Documentation	92
7.13.2.1 available_profiles	92
7.13.2.2 current	93
7.13.2.3 num_profiles	93
7.14 rsmi_process_info_t Struct Reference	93
7.14.1 Detailed Description	93
7.15 rsmi_range_t Struct Reference	94
7.15.1 Detailed Description	94
7.16 rsmi_retired_page_record_t Struct Reference	94
7.16.1 Detailed Description	94
7.17 rsmi_utilization_counter_t Struct Reference	95
7.17.1 Detailed Description	95
7.18 rsmi_version_t Struct Reference	95
7.18.1 Detailed Description	95
<b>8 File Documentation</b>	<b>97</b>
8.1 rocm_smi.h File Reference	97
8.1.1 Detailed Description	106
8.1.2 Macro Definition Documentation	106
8.1.2.1 RSMI_MAX_FAN_SPEED	106
8.1.2.2 RSMI_EVENT_MASK_FROM_INDEX	106
8.1.2.3 RSMI_DEFAULT_VARIANT	107
8.1.3 Typedef Documentation	107
8.1.3.1 rsmi_event_handle_t	107
8.1.4 Enumeration Type Documentation	107
8.1.4.1 rsmi_status_t	107
8.1.4.2 rsmi_init_flags_t	108
8.1.4.3 rsmi_dev_perf_level_t	108

8.1.4.4 rsmi_sw_component_t . . . . .	108
8.1.4.5 rsmi_event_group_t . . . . .	109
8.1.4.6 rsmi_event_type_t . . . . .	109
8.1.4.7 rsmi_counter_command_t . . . . .	109
8.1.4.8 rsmi_evt_notification_type_t . . . . .	110
8.1.4.9 rsmi_clk_type_t . . . . .	110
8.1.4.10 rsmi_compute_partition_type_t . . . . .	110
8.1.4.11 rsmi_nps_mode_type_t . . . . .	111
8.1.4.12 rsmi_temperature_metric_t . . . . .	111
8.1.4.13 rsmi_temperature_type_t . . . . .	112
8.1.4.14 rsmi_voltage_metric_t . . . . .	112
8.1.4.15 rsmi_voltage_type_t . . . . .	112
8.1.4.16 rsmi_power_profile_preset_masks_t . . . . .	113
8.1.4.17 rsmi_gpu_block_t . . . . .	113
8.1.4.18 rsmi_ras_err_state_t . . . . .	113
8.1.4.19 rsmi_memory_type_t . . . . .	114
8.1.4.20 rsmi_freq_ind_t . . . . .	114
8.1.4.21 rsmi_memory_page_status_t . . . . .	114
8.1.4.22 _RSMI_IO_LINK_TYPE . . . . .	115
8.1.4.23 RSMI_UTILIZATION_COUNTER_TYPE . . . . .	115
8.2 rocm_smi.h . . . . .	115
<b>Index</b>	<b>127</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.

### 1.1 DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. In addition, any stated support is planned and is also subject to change. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein.

© 2022 Advanced Micro Devices, Inc. All Rights Reserved.

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

```
$ mkdir -p build
```

```
$ cd build
```

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

```
$ make
```

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

```
$ make install
```

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

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

```
$ make package
```

## Documentation

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

```
$ mkdir <location for test build>
```

```
$ cd <location for test build>
```

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

```
$ make
```

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

## 1.3 Usage Basics

### 1.3.1 Device Indices

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

## 1.4 Hello ROCm SMI

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

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

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

```
#include <stdint.h>
#include "rocm_smi/rocm_smi.h"
int main() {
    rsmi_status_t ret;
    uint32_t num_devices;
    uint16_t dev_id;

    // We will skip return code checks for this example, but it
    // is recommended to always check this as some calls may not
    // apply for some devices or ROCm releases

    ret = rsmi_init(0);
    ret = rsmi_num_monitor_devices(&num_devices);

    for (int i=0; i < num_devices; ++i) {
        ret = rsmi_dev_id_get(i, &dev_id);
        // dev_id holds the device ID of device i, upon a
        // successful call
    }
    ret = rsmi_shut_down();
    return 0;
}
```



## Chapter 2

# Deprecated List

Global [rsmi\\_dev\\_overdrive\\_level\\_set](#) (int32\_t dv\_ind, uint32\_t od)

This function is deprecated. [rsmi\\_dev\\_overdrive\\_level\\_set\\_v1](#) has the same functionality, with an interface that more closely matches the rest of the rocm\_smi API.

Global [rsmi\\_dev\\_perf\\_level\\_set](#) (int32\_t dv\_ind, rsmi\_dev\_perf\_level\_t perf\_lvl)

[rsmi\\_dev\\_perf\\_level\\_set\\_v1\(\)](#) is preferred, with an interface that more closely matches the rest of the rocm\_smi API.



## Chapter 3

# Module Index

### 3.1 Modules

Here is a list of all modules:

Initialization and Shutdown . . . . .	13
Identifier Queries . . . . .	14
PCIe Queries . . . . .	22
PCIe Control . . . . .	26
Power Queries . . . . .	27
Power Control . . . . .	30
Memory Queries . . . . .	32
Physical State Queries . . . . .	35
Physical State Control . . . . .	38
Clock, Power and Performance Queries . . . . .	40
Clock, Power and Performance Control . . . . .	49
Version Queries . . . . .	53
Error Queries . . . . .	56
Performance Counter Functions . . . . .	58
System Information Functions . . . . .	63
XGMI Functions . . . . .	65
Hardware Topology Functions . . . . .	67
Compute Partition Functions . . . . .	71
NPS Mode Functions . . . . .	74
Supported Functions . . . . .	76
Event Notification Functions . . . . .	81



## Chapter 4

# Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">id</a>	This union holds the value of an <a href="#">rsmi_func_id_iter_handle_t</a> . The value may be a function name, or an enumerated variant value of types such as <a href="#">rsmi_memory_type_t</a> , <a href="#">rsmi_temperature_metric_t</a> , etc . . . . .	85
<a href="#">metrics_table_header_t</a>	The following structures hold the gpu metrics values for a device . . . . .	86
<a href="#">rsmi_counter_value_t</a>		86
<a href="#">rsmi_error_count_t</a>	This structure holds error counts . . . . .	87
<a href="#">rsmi_evt_notification_data_t</a>		87
<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> . . . . .	88
<a href="#">rsmi_frequencies_t</a>	This structure holds information about clock frequencies . . . . .	88
<a href="#">rsmi_gpu_metrics_t</a>		89
<a href="#">rsmi_od_vddc_point_t</a>	This structure represents a point on the frequency-voltage plane . . . . .	89
<a href="#">rsmi_od_volt_curve_t</a>		90
<a href="#">rsmi_od_volt_freq_data_t</a>	This structure holds the frequency-voltage values for a device . . . . .	90
<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 . . . . .	91
<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 . . . . .	92
<a href="#">rsmi_process_info_t</a>	This structure contains information specific to a process . . . . .	93
<a href="#">rsmi_range_t</a>	This structure represents a range (e.g., frequencies or voltages) . . . . .	94
<a href="#">rsmi_retired_page_record_t</a>	Reserved Memory Page Record . . . . .	94
<a href="#">rsmi_utilization_counter_t</a>	The utilization counter data . . . . .	95
<a href="#">rsmi_version_t</a>	This structure holds version information . . . . .	95



# Chapter 5

## File Index

### 5.1 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 . . . . .

97



# Chapter 6

## Module Documentation

### 6.1 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.*

#### 6.1.1 Detailed Description

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

#### 6.1.2 Function Documentation

##### 6.1.2.1 rsmi\_init()

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

Initialize ROCm SMI.

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

#### Parameters

in	<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.
-------------------------------------	-----------------------------------

### 6.1.2.2 rsmi\_shut\_down()

```
rsmi_status_t rsmi_shut_down (
    void )
```

Shutdown ROCm SMI.

Do any necessary clean up.

## 6.2 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\\_sku\\_get](#) (uint32\_t dv\_ind, char \*sku)  
*Get the SKU for a desired device associated with the device with provided device index.*
- [rsmi\\_status\\_t rsmi\\_dev\\_vendor\\_id\\_get](#) (uint32\_t dv\_ind, uint16\_t \*id)  
*Get the device vendor id associated with the device with provided device index.*
- [rsmi\\_status\\_t rsmi\\_dev\\_name\\_get](#) (uint32\_t dv\_ind, char \*name, size\_t len)  
*Get the name string of a gpu device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_brand\\_get](#) (uint32\_t dv\_ind, char \*brand, uint32\_t len)  
*Get the brand string of a gpu device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_vendor\\_name\\_get](#) (uint32\_t dv\_ind, char \*name, size\_t len)  
*Get the name string for a give vendor ID.*
- [rsmi\\_status\\_t rsmi\\_dev\\_vram\\_vendor\\_get](#) (uint32\_t dv\_ind, char \*brand, uint32\_t len)  
*Get the vram vendor string of a gpu device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_serial\\_number\\_get](#) (uint32\_t dv\_ind, char \*serial\_num, uint32\_t len)  
*Get the serial number string for a device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_subsystem\\_id\\_get](#) (uint32\_t dv\_ind, uint16\_t \*id)  
*Get the subsystem device id associated with the device with provided device index.*
- [rsmi\\_status\\_t rsmi\\_dev\\_subsystem\\_name\\_get](#) (uint32\_t dv\_ind, char \*name, size\_t len)  
*Get the name string for the device subsytem.*
- [rsmi\\_status\\_t rsmi\\_dev\\_drm\\_render\\_minor\\_get](#) (uint32\_t dv\_ind, uint32\_t \*minor)  
*Get the drm minor number associated with this device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_subsystem\\_vendor\\_id\\_get](#) (uint32\_t dv\_ind, uint16\_t \*id)  
*Get the device subsystem vendor id associated with the device with provided device index.*
- [rsmi\\_status\\_t rsmi\\_dev\\_unique\\_id\\_get](#) (uint32\_t dv\_ind, uint64\_t \*id)  
*Get Unique ID.*

### 6.2.1 Detailed Description

These functions provide identification information.

## 6.2.2 Function Documentation

### 6.2.2.1 `rsmi_num_monitor_devices()`

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

Get the number of devices that have monitor information.

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

#### Parameters

<code>in, out</code>	<code>num_devices</code>	Caller provided pointer to <code>uint32_t</code> . Upon successful call, the value <code>num_devices</code> will contain the number of monitor devices.
----------------------	--------------------------	---

#### Return values

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

### 6.2.2.2 `rsmi_dev_id_get()`

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

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

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

#### Parameters

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

#### Return values

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

### 6.2.2.3 rsmi\_dev\_sku\_get()

```
rsmi_status_t rsmi_dev_sku_get (
    uint32_t dv_ind,
    char * sku )
```

Get the SKU for a desired device associated with the device with provided device index.

Given a device index `dv_ind` and a pointer to a char `sku`, this function will attempt to obtain the SKU from the Product Information FRU chip, present on server ASICs. It will write the sku value to the char array pointed to by `sku`.

#### Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>sku</i>	a pointer to char to which the sku will be written

If this parameter is nullptr, this function will return [RSMI\\_STATUS\\_INVALID\\_ARGS](#) if the function is supported with the provided, arguments and [RSMI\\_STATUS\\_NOT\\_SUPPORTED](#) if it is not supported with the provided arguments.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

### 6.2.2.4 rsmi\_dev\_vendor\_id\_get()

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

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

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

#### Parameters

in	<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 If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided, arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments

## Return values

<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
---	--------------------------------------

6.2.2.5 `rsmi_dev_name_get()`

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

Get the name string of a gpu device.

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

If the integer ID associated with the device is not found in one of the system files containing device name information (e.g. `/usr/share/misc/pci.ids`), then this function will return the hex device ID as a string. Updating the system name files can be 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. If this parameter is nullptr, this function will return <a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a> if the function is supported with the provided arguments and <a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a> if it is not supported with the provided arguments.
in	<i>len</i>	the length of the caller provided buffer <code>name</code> .

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<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.

6.2.2.6 `rsmi_dev_brand_get()`

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

Get the brand string of a gpu device.

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

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

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>brand</i>	a pointer to a caller provided char buffer to which the brand will be written. If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.
in	<i>len</i>	the length of the caller provided buffer <i>brand</i> .

## Return values

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

6.2.2.7 `rsmi_dev_vendor_name_get()`

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

Get the name string for a give vendor ID.

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

If the integer ID associated with the vendor is not found in one of the system files containing device name information (e.g. `/usr/share/misc/pci.ids`), then this function will return the hex vendor ID as a string. Updating the system name files can be 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. If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.
in	<i>len</i>	the length of the caller provided buffer <i>name</i> .

## Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

## Return values

<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.
--	---

6.2.2.8 `rsmi_dev_vram_vendor_get()`

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

Get the vram vendor string of a gpu device.

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

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

## Parameters

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

## Return values

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

6.2.2.9 `rsmi_dev_serial_number_get()`

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

Get the serial number string for a device.

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

## Parameters

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

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<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.

6.2.2.10 `rsmi_dev_subsystem_id_get()`

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

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

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

## Parameters

<code>in</code>	<code>dv_ind</code>	a device index
<code>in, out</code>	<code>id</code>	a pointer to <code>uint64_t</code> to which the subsystem device id will be written. If this parameter is <code>nullptr</code> , this function will return <a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a> if the function is supported with the provided arguments and <a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a> if it is not supported with the provided arguments.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

6.2.2.11 `rsmi_dev_subsystem_name_get()`

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

Get the name string for the device 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. If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.
in	<i>len</i>	the length of the caller provided buffer <i>name</i> .

## Return values

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

6.2.2.12 `rsmi_dev_drm_render_minor_get()`

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

Get the drm minor number associated with this device.

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

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>minor</i>	a pointer to a <code>uint32_t</code> into which minor number will be copied

## Return values

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

6.2.2.13 `rsmi_dev_subsystem_vendor_id_get()`

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

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

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

## Parameters

in	<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. If this parameter is <code>nullptr</code> , this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

## Return values

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

6.2.2.14 `rsmi_dev_unique_id_get()`

```
rsmi_status_t rsmi_dev_unique_id_get (
    uint32_t dv_ind,
    uint64_t * id )
```

Get Unique ID.

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

## Parameters

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

## Return values

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

## 6.3 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\\_topo\\_numa\\_affinity\\_get](#) (uint32\_t dv\_ind, uint32\_t \*numa\_node)  
*Get the NUMA node associated with a device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_pci\\_throughput\\_get](#) (uint32\_t dv\_ind, uint64\_t \*sent, uint64\_t \*received, uint64\_t \*max\_pkt\_sz)  
*Get PCIe traffic information.*
- [rsmi\\_status\\_t rsmi\\_dev\\_pci\\_replay\\_counter\\_get](#) (uint32\_t dv\_ind, uint64\_t \*counter)  
*Get PCIe replay counter.*

### 6.3.1 Detailed Description

These functions provide information about PCIe.

### 6.3.2 Function Documentation

#### 6.3.2.1 rsmi\_dev\_pci\_bandwidth\_get()

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

Get the list of possible PCIe bandwidths that are available.

Given a device index `dv_ind` and a pointer to a `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.
-------------------------------------	-----------------------------------

#### 6.3.2.2 rsmi\_dev\_pci\_id\_get()

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

Get the unique PCI device identifier associated for a device.

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

The format of `bdfid` will be as follows:

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

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

#### Parameters

in	<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. If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

### 6.3.2.3 rsmi\_topo\_numa\_affinity\_get()

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

Get the NUMA node associated with a device.

Given a device index *dv\_ind* and a pointer to a uint32\_t *numa\_node*, this function will retrieve the NUMA node value associated with device *dv\_ind* and store the value at location pointed to by *numa\_node*.

#### Parameters

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

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments

## Return values

<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
---	--------------------------------------

6.3.2.4 `rsmi_dev_pci_throughput_get()`

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

Get PCIe traffic information.

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

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>sent</i>	a pointer to <code>uint64_t</code> 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 <code>uint64_t</code> 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 <code>uint64_t</code> 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.
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments

6.3.2.5 `rsmi_dev_pci_replay_counter_get()`

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

Get PCIe replay counter.

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

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>counter</i>	a pointer to <code>uint64_t</code> to which the sum of the NAK's received and generated by the GPU is written. If this parameter is nullptr, this function will return <a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a> if the function is supported with the provided arguments and <a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a> if it is not supported with the provided arguments.
Generated by Doxygen		

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

## 6.4 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.*

#### 6.4.1 Detailed Description

These functions provide some control over PCIe.

#### 6.4.2 Function Documentation

##### 6.4.2.1 `rsmi_dev_pci_bandwidth_set()`

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

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

Given a device index `dv_ind` and a 64 bit bitmask `bw_bitmask`, this function will limit the set of allowable bandwidths. If a bit in `bw_bitmask` has a value of 1, then the frequency (as ordered in an [`rsmi\_frequencies\_t`](#) returned by [`rsmi\_dev\_gpu\_clk\_freq\_get\(\)`](#)) corresponding to that bit index will be allowed.

This function will change the performance level to [`RSMI\_DEV\_PERF\_LEVEL\_MANUAL`](#) in order to modify the set of allowable band\_widths. Caller will need to set to [`RSMI\_DEV\_PERF\_LEVEL\_AUTO`](#) in order to get back to default state.

All bits with indices greater than or equal to the value of the [`rsmi\_frequencies\_t::num\_supported`](#) field of [`rsmi\_pcie\_bandwidth\_t`](#) will be ignored.

### Parameters

in	<i><code>dv_ind</code></i>	a device index
in	<i><code>bw_bitmask</code></i>	A bitmask indicating the indices of the bandwidths that are to be enabled (1) and disabled (0). Only the lowest <a href="#"><code>rsmi_frequencies_t::num_supported</code></a> (of <a href="#"><code>rsmi_pcie_bandwidth_t</code></a> ) bits of this mask are relevant.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_PERMISSION</i></a>	function requires root access

## 6.5 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\_energy\_count\_get`](#) (uint32\_t dv\_ind, uint64\_t \*power, float \*counter\_resolution, uint64\_t \*timestamp)  
*Get the energy accumulator counter 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\_default\_get`](#) (uint32\_t dv\_ind, uint64\_t \*default\_cap)  
*Get the default power cap for the device specified by dv\_ind.*
- [`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.*

### 6.5.1 Detailed Description

These functions provide information about power usage.

### 6.5.2 Function Documentation

#### 6.5.2.1 `rsmi_dev_power_ave_get()`

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

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

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

#### Parameters

in	<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. If this parameter is nullptr, this function will return <a href="#"><code>RSMI_STATUS_INVALID_ARGS</code></a> if the function is supported with the provided arguments and <a href="#"><code>RSMI_STATUS_NOT_SUPPORTED</code></a> if it is not supported with the provided arguments.
Generated by Doxygen		

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

**6.5.2.2 rsmi\_dev\_energy\_count\_get()**

```
rsmi_status_t rsmi_dev_energy_count_get (
    uint32_t dv_ind,
    uint64_t * power,
    float * counter_resolution,
    uint64_t * timestamp )
```

Get the energy accumulator counter of the device with provided device index.

Given a device index `dv_ind`, a pointer to a `uint64_t` `power`, and a pointer to a `uint64_t` `timestamp`, this function will write amount of energy consumed to the `uint64_t` pointed to by `power`, and the timestamp to the `uint64_t` pointed to by `timestamp`. The [`rsmi\_dev\_power\_ave\_get\(\)`](#) is an average of a short time. This function accumulates all energy consumed.

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>counter_resolution</i>	resolution of the counter <code>power</code> in micro Joules
in, out	<i>power</i>	a pointer to <code>uint64_t</code> to which the energy counter will be written. If this parameter is nullptr, this function will return <a href="#"><code>RSMI_STATUS_INVALID_ARGS</code></a> if the function is supported with the provided, and <a href="#"><code>RSMI_STATUS_NOT_SUPPORTED</code></a> if it is not supported with the provided arguments.
in, out	<i>timestamp</i>	a pointer to <code>uint64_t</code> to which the timestamp will be written. Resolution: 1 ns.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

**6.5.2.3 rsmi\_dev\_power\_cap\_get()**

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

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

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

## Parameters

in	<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 pointer to a <code>uint64_t</code> that indicates the power cap, in microwatts. If this parameter is <code>nullptr</code> , this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

## Return values

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

6.5.2.4 `rsmi_dev_power_cap_default_get()`

```
rsmi_status_t rsmi_dev_power_cap_default_get (
    uint32_t dv_ind,
    uint64_t * default_cap )
```

Get the default power cap for the device specified by `dv_ind`.

The maximum power cap can be temporarily changed by the user. However, this function always returns the default reset power cap. The power level returned through `power` will be in microWatts.

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>default_cap</i>	a pointer to a <code>uint64_t</code> that indicates the default power cap, in microwatts. If this parameter is <code>nullptr</code> , this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

## Return values

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

6.5.2.5 `rsmi_dev_power_cap_range_get()`

```
rsmi_status_t rsmi_dev_power_cap_range_get (
    uint32_t dv_ind,
```

```
uint32_t sensor_ind,
uint64_t * max,
uint64_t * min )
```

Get the range of valid values for the power cap.

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

#### Parameters

in	<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 If this parameter is <code>nullptr</code> , this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.
in, out	<i>min</i>	a pointer to a <code>uint64_t</code> that indicates the minimum possible power cap, in microwatts If this parameter is <code>nullptr</code> , this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

## 6.6 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 reserved`, [rsmi\\_power\\_profile\\_preset\\_masks\\_t](#) profile)  
*Set the power profile.*

### 6.6.1 Detailed Description

These functions provide ways to control power usage.

## 6.6.2 Function Documentation

### 6.6.2.1 `rsmi_dev_power_cap_set()`

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

Set the power cap value.

This function will set the power cap to the provided value `cap`. `cap` must be between the minimum and maximum power cap values set by the system, which can be obtained from [rsmi\\_dev\\_power\\_cap\\_range\\_get](#).

#### Parameters

in	<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>cap</i>	a <code>uint64_t</code> that indicates the desired power cap, in microwatts

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	is returned upon successful call.
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid
<a href="#">RSMI_STATUS_PERMISSION</a>	function requires root access

### 6.6.2.2 `rsmi_dev_power_profile_set()`

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

Set the power profile.

Given a device index `dv_ind` and a `profile`, this function will attempt to set the current profile to the provided profile. The provided profile must be one of the currently supported profiles, as indicated by a call to [rsmi\\_dev\\_power\\_profile\\_presets\\_get\(\)](#)

#### Parameters

in	<i>dv_ind</i>	a device index
in	<i>reserved</i>	Not currently used. Set to 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

<a href="#">RSMI_STATUS_SUCCESS</a>	is returned upon successful call.
<a href="#">RSMI_STATUS_PERMISSION</a>	function requires root access

## 6.7 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.*
- [rsmi\\_status\\_t rsmi\\_dev\\_memory\\_reserved\\_pages\\_get](#) (uint32\_t dv\_ind, uint32\_t \*num\_pages, [rsmi\\_retired\\_page\\_record\\_t](#) \*records)  
*Get information about reserved ("retired") memory pages.*

### 6.7.1 Detailed Description

These functions provide information about memory systems.

### 6.7.2 Function Documentation

#### 6.7.2.1 rsmi\_dev\_memory\_total\_get()

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

Get the total amount of memory that exists.

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

#### Parameters

in	<i>dv_ind</i>	a device index
in	<i>mem_type</i>	The type of memory for which the total amount will be found
in, out	<i>total</i>	a pointer to <code>uint64_t</code> to which the total amount of memory will be written If this parameter is <code>nullptr</code> , this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

### 6.7.2.2 `rsmi_dev_memory_usage_get()`

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

Get the current memory usage.

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

#### Parameters

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

#### Return values

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

### 6.7.2.3 `rsmi_dev_memory_busy_percent_get()`

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

Get percentage of time any device memory is being used.

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

#### Parameters

in	<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. If this parameter is <code>nullptr</code> , this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

6.7.2.4 `rsmi_dev_memory_reserved_pages_get()`

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

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

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

## Parameters

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

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<a href="#"><i>RSMI_STATUS_INSUFFICIENT_SIZE</i></a>	is returned if more records were available than allowed by the provided, allocated memory.

## 6.8 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.*
- [rsmi\\_status\\_t rsmi\\_dev\\_volt\\_metric\\_get](#) (uint32\_t dv\_ind, [rsmi\\_voltage\\_type\\_t](#) sensor\_type, [rsmi\\_voltage\\_metric\\_t](#) metric, int64\_t \*voltage)  
*Get the voltage metric value for the specified metric, from the specified voltage sensor on the specified device.*

### 6.8.1 Detailed Description

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

### 6.8.2 Function Documentation

#### 6.8.2.1 rsmi\_dev\_fan\_rpms\_get()

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

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

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

#### Parameters

in	<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 <code>uint32_t</code> to which the speed will be written If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
-------------------------------------	---------------------

## Return values

<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

**6.8.2.2 rsmi\_dev\_fan\_speed\_get()**

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

Get the fan speed for the specified device as a value relative to [\*RSMI\\_MAX\\_FAN\\_SPEED\*](#).

Given a device index `dv_ind` and a pointer to a `uint32_t` `speed`, this function will write the current fan speed (a value between 0 and the maximum fan speed, [\*RSMI\\_MAX\\_FAN\\_SPEED\*](#)) to the `uint32_t` pointed to by `speed`

## Parameters

in	<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 <code>uint32_t</code> to which the speed will be written. If this parameter is nullptr, this function will return <a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a> if the function is supported with the provided arguments and <a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a> if it is not supported with the provided arguments.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

**6.8.2.3 rsmi\_dev\_fan\_speed\_max\_get()**

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

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

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

## Parameters

in	<i>dv_ind</i>	a device index
----	---------------	----------------

## Parameters

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 <code>uint32_t</code> to which the maximum speed will be written. If this parameter is <code>nullptr</code> , this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

## Return values

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

6.8.2.4 `rsmi_dev_temp_metric_get()`

```

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

```

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

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

## Parameters

in	<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 <code>rsmi_temperature_type_t</code>
in	<i>metric</i>	enum indicated which temperature value should be retrieved
in, out	<i>temperature</i>	a pointer to <code>int64_t</code> to which the temperature will be written, in millidegrees Celcius. If this parameter is <code>nullptr</code> , this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

## Return values

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

### 6.8.2.5 `rsmi_dev_volt_metric_get()`

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

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

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

#### Parameters

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

#### Return values

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

## 6.9 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.*

### 6.9.1 Detailed Description

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

## 6.9.2 Function Documentation

### 6.9.2.1 rsmi\_dev\_fan\_reset()

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

Reset the fan to automatic driver control.

This function returns control of the fan to the system

#### Parameters

in	<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

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

### 6.9.2.2 rsmi\_dev\_fan\_speed\_set()

```
rsmi_status_t rsmi_dev_fan_speed_set (
    uint32_t dv_ind,
    uint32_t sensor_ind,
    uint64_t speed )
```

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

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

#### Parameters

in	<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.
<i>RSMI_STATUS_NOT_SUPPORTED</i>	installed software or hardware does not support this function with the given arguments
<i>RSMI_STATUS_PERMISSION</i>	function requires root access

## 6.10 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\\_utilization\\_count\\_get](#) (uint32\_t dv\_ind, [rsmi\\_utilization\\_counter\\_t](#) utilization\_counters[], uint32\_t count, uint64\_t \*timestamp)  
*Get coarse grain utilization counter of the specified device.*
- [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\\_perf\\_determinism\\_mode\\_set](#) (uint32\_t dv\_ind, uint64\_t clkvalue)  
*Enter performance determinism mode 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\\_mem\\_overdrive\\_level\\_get](#) (uint32\_t dv\_ind, uint32\_t \*od)  
*Get the memory clock 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\\_gpu\\_reset](#) (uint32\_t dv\_ind)  
*Reset the gpu associated with the device with provided device index.*
- [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\\_gpu\\_metrics\\_info\\_get](#) (uint32\_t dv\_ind, [rsmi\\_gpu\\_metrics\\_t](#) \*pgpu\_metrics)  
*This function retrieves the gpu metrics information.*
- [rsmi\\_status\\_t rsmi\\_dev\\_clk\\_range\\_set](#) (uint32\_t dv\_ind, uint64\_t minclkvalue, uint64\_t maxclkvalue, [rsmi\\_clk\\_type\\_t](#) clkType)  
*This function sets the clock range information.*
- [rsmi\\_status\\_t rsmi\\_dev\\_od\\_clk\\_info\\_set](#) (uint32\_t dv\_ind, [rsmi\\_freq\\_ind\\_t](#) level, uint64\_t clkvalue, [rsmi\\_clk\\_type\\_t](#) clkType)  
*This function sets the clock frequency information.*
- [rsmi\\_status\\_t rsmi\\_dev\\_od\\_volt\\_info\\_set](#) (uint32\_t dv\_ind, uint32\_t vpoint, uint64\_t clkvalue, uint64\_t volt-value)  
*This function sets 1 of the 3 voltage curve points.*
- [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.*

### 6.10.1 Detailed Description

These functions provide information about clock frequencies and performance.

## 6.10.2 Function Documentation

### 6.10.2.1 `rsmi_dev_busy_percent_get()`

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

Get percentage of time device is busy doing any processing.

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

#### Parameters

in	<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. If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Return values

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

### 6.10.2.2 `rsmi_utilization_count_get()`

```
rsmi_status_t rsmi_utilization_count_get (
    uint32_t dv_ind,
    rsmi_utilization_counter_t utilization_counters[],
    uint32_t count,
    uint64_t * timestamp )
```

Get coarse grain utilization counter of the specified device.

Given a device index `dv_ind`, the array of the utilization counters, the size of the array, this function returns the coarse grain utilization counters and timestamp. The counter is the accumulated percentages. Every milliseconds the firmware calculates % busy count and then accumulates that value in the counter. This provides minimally invasive coarse grain GPU usage information.

#### Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>utilization_counters</i>	Multiple utilization counters can be retrieved with a single call. The caller must allocate enough space to the <code>utilization_counters</code> array. The caller also needs to set valid <code>RSMI_UTILIZATION_COUNTER_TYPE</code> type for each element of the array. <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

If the function returns `RSMI_STATUS_SUCCESS`, the counter will be set in the value field of the [rsmi\\_utilization\\_counter\\_t](#).

#### Parameters

in	<i>count</i>	The size of @utilization_counters array.
in, out	<i>timestamp</i>	The timestamp when the counter is retrieved. Resolution: 1 ns.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

### 6.10.2.3 rsmi\_dev\_perf\_level\_get()

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

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

Given a device index `dv_ind` and a pointer to a `uint32_t perf`, this function will write the [rsmi\\_dev\\_perf\\_level\\_t](#) to the `uint32_t` pointed to by `perf`

#### Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>perf</i>	a pointer to <a href="#">rsmi_dev_perf_level_t</a> to which the performance level will be written. If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

### 6.10.2.4 rsmi\_perf\_determinism\_mode\_set()

```
rsmi_status_t rsmi_perf_determinism_mode_set (
    uint32_t dv_ind,
    uint64_t clkvalue )
```

Enter performance determinism mode with provided device index.

Given a device index `dv_ind` and `clkvalue` this function will enable performance determinism mode, which enforces a GFXCLK frequency SoftMax limit per GPU set by the user. This prevents the GFXCLK PLL from stretching when running the same workload on different GPUS, making performance variation minimal. This call will result in the performance level `rsmi_dev_perf_level_t` of the device being `RSMI_DEV_PERF_LEVEL_DETERMINISM`.

## Parameters

in	<i>dv_ind</i>	a device index
in	<i>clkvalue</i>	Softmax value for GFXCLK in MHz.

## Return values

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

6.10.2.5 `rsmi_dev_overdrive_level_get()`

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

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

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

## Parameters

in	<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. If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

## Return values

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

6.10.2.6 `rsmi_dev_mem_overdrive_level_get()`

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

Get the memory clock 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 memory 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. If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Return values

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

### 6.10.2.7 `rsmi_dev_gpu_clk_freq_get()`

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

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

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

#### Parameters

in	<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. If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments. If multiple current frequencies are found, a warning is shown. If no current frequency is found, it is reflected as -1. If frequencies are not read from low to high a warning is shown as well.

#### Return values

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

### 6.10.2.8 rsmi\_dev\_gpu\_reset()

```
rsmi_status_t rsmi_dev_gpu_reset (
    int32_t dv_ind )
```

Reset the gpu associated with the device with provided device index.

Given a device index `dv_ind`, this function will reset the GPU

#### Parameters

in	<i>dv_ind</i>	a device index
----	---------------	----------------

#### Return values

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

### 6.10.2.9 rsmi\_dev\_od\_volt\_info\_get()

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

This function retrieves the voltage/frequency curve information.

Given a device index `dv_ind` and a pointer to a `rsmi_od_volt_freq_data_t` structure `odv`, this function will populate `odv`. See [rsmi\\_od\\_volt\\_freq\\_data\\_t](#) for more details.

#### Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>odv</i>	a pointer to an <a href="#">rsmi_od_volt_freq_data_t</a> structure If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided, arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Return values

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

### 6.10.2.10 rsmi\_dev\_gpu\_metrics\_info\_get()

```
rsmi_status_t rsmi_dev_gpu_metrics_info_get (
```

```
uint32_t dv_ind,
rsmi_gpu_metrics_t * pgpu_metrics )
```

This function retrieves the gpu metrics information.

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

#### Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>pgpu_metrics</i>	a pointer to an <code>rsmi_gpu_metrics_t</code> structure If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided, arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Return values

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

#### 6.10.2.11 rsmi\_dev\_clk\_range\_set()

```
rsmi_status_t rsmi_dev_clk_range_set (
    uint32_t dv_ind,
    uint64_t minclkvalue,
    uint64_t maxclkvalue,
    rsmi_clk_type_t clkType )
```

This function sets the clock range information.

Given a device index `dv_ind`, a minimum clock value `minclkvalue`, a maximum clock value `maxclkvalue` and a clock type `clkType` this function will set the sclk|mclk range

#### Parameters

in	<i>dv_ind</i>	a device index
in	<i>minclkvalue</i>	value to apply to the clock range. Frequency values are in MHz.
in	<i>maxclkvalue</i>	value to apply to the clock range. Frequency values are in MHz.
in	<i>clkType</i>	<code>RSMI_CLK_TYPE_SYS</code>   <code>RSMI_CLK_TYPE_MEM</code> range type

#### Return values

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

6.10.2.12 `rsmi_dev_od_clk_info_set()`

```
rsmi_status_t rsmi_dev_od_clk_info_set (
    uint32_t dv_ind,
    rsmi_freq_ind_t level,
    uint64_t clkvalue,
    rsmi_clk_type_t clkType )
```

This function sets the clock frequency information.

Given a device index `dv_ind`, a frequency level `level`, a clock value `clkvalue` and a clock type `clkType` this function will set the `sclk|mclk` range

## Parameters

in	<i>dv_ind</i>	a device index
in	<i>level</i>	RSMI_FREQ_IND_MIN RSMI_FREQ_IND_MAX to set the minimum (0) or maximum (1) speed.
in	<i>clkvalue</i>	value to apply to the clock range. Frequency values are in MHz.
in	<i>clkType</i>	RSMI_CLK_TYPE_SYS   RSMI_CLK_TYPE_MEM range type

## Return values

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

6.10.2.13 `rsmi_dev_od_volt_info_set()`

```
rsmi_status_t rsmi_dev_od_volt_info_set (
    uint32_t dv_ind,
    uint32_t vpoint,
    uint64_t clkvalue,
    uint64_t voltvalue )
```

This function sets 1 of the 3 voltage curve points.

Given a device index `dv_ind`, a voltage point `vpoint` and a voltage value `voltvalue` this function will set voltage curve point

## Parameters

in	<i>dv_ind</i>	a device index
in	<i>vpoint</i>	voltage point [0 1 2] on the voltage curve
in	<i>clkvalue</i>	clock value component of voltage curve point. Frequency values are in MHz.
in	<i>voltvalue</i>	voltage value component of voltage curve point. Voltage is in mV.

## Return values

<i>RSMI_STATUS_SUCCESS</i>	call was successful
----------------------------	---------------------

## Return values

<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

**6.10.2.14 rsmi\_dev\_od\_volt\_curve\_regions\_get()**

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

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

Given a device index `dv_ind`, a pointer to an unsigned integer `num_regions` and a buffer of [`rsmi\_freq\_volt\_region\_t`](#) structures, `buffer`, this function will populate `buffer` with the current frequency-volt space regions. The caller should assign `buffer` to memory that can be written to by this function. The caller should also indicate the number of [`rsmi\_freq\_volt\_region\_t`](#) structures that can safely be written to `buffer` in `num_regions`.

The number of regions to expect this function provide (`num_regions`) can be obtained by calling [`rsmi\_dev\_od\_volt\_info\_get\(\)`](#).

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>num_regions</i>	As input, this is the number of <a href="#"><code>rsmi_freq_volt_region_t</code></a> structures that can be written to <code>buffer</code> . As output, this is the number of <a href="#"><code>rsmi_freq_volt_region_t</code></a> structures that were actually written. If this parameter is nullptr, this function will return <a href="#"><code>RSMI_STATUS_INVALID_ARGS</code></a> if the function is supported with the provided arguments and <a href="#"><code>RSMI_STATUS_NOT_SUPPORTED</code></a> if it is not supported with the provided arguments.
in, out	<i>buffer</i>	a caller provided buffer to which <a href="#"><code>rsmi_freq_volt_region_t</code></a> structures will be written. If this parameter is nullptr, this function will return <a href="#"><code>RSMI_STATUS_INVALID_ARGS</code></a> if the function is supported with the provided arguments and <a href="#"><code>RSMI_STATUS_NOT_SUPPORTED</code></a> if it is not supported with the provided arguments.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

**6.10.2.15 rsmi\_dev\_power\_profile\_presets\_get()**

```
rsmi_status_t rsmi_dev_power_profile_presets_get (
    uint32_t dv_ind,
```

```
uint32_t sensor_ind,
rsmi_power_profile_status_t * status )
```

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

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

#### Parameters

in	<code>dv_ind</code>	a device index
in	<code>sensor_ind</code>	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	<code>status</code>	a pointer to <code>rsmi_power_profile_status_t</code> that will be populated by a call to this function. If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Return values

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

## 6.11 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_perf_level_set_v1` (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` (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_overdrive_level_set_v1` (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.

### 6.11.1 Detailed Description

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

## 6.11.2 Function Documentation

### 6.11.2.1 `rsmi_dev_perf_level_set()`

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

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

**Deprecated** `rsmi_dev_perf_level_set_v1()` is preferred, with an interface that more closely matches the rest of the rocm\_smi API.

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	<i>dv_ind</i>	a device index
in	<i>perf_↔ _lvl</i>	the value to which the performance level should be set

#### Return values

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

### 6.11.2.2 `rsmi_dev_perf_level_set_v1()`

```
rsmi_status_t rsmi_dev_perf_level_set_v1 (
    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.

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	<i>dv_ind</i>	a device index
in	<i>perf_↔ _lvl</i>	the value to which the performance level should be set

#### Return values

<i>RSMI_STATUS_SUCCESS</i>	is returned upon successful call.
----------------------------	-----------------------------------

## Return values

<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_PERMISSION</i></a>	function requires root access

6.11.2.3 `rsmi_dev_overdrive_level_set()`

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

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

**Deprecated** This function is deprecated. [`rsmi\_dev\_overdrive\_level\_set\_v1`](#) has the same functionality, with an interface that more closely matches the rest of the rocm\_smi API.

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>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_PERMISSION</i></a>	function requires root access

#### 6.11.2.4 `rsmi_dev_overdrive_level_set_v1()`

```
rsmi_status_t rsmi_dev_overdrive_level_set_v1 (
    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.

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

<i><a href="#">RSMI_STATUS_SUCCESS</a></i>	call was successful
<i><a href="#">RSMI_STATUS_NOT_SUPPORTED</a></i>	installed software or hardware does not support this function with the given arguments
<i><a href="#">RSMI_STATUS_PERMISSION</a></i>	function requires root access

#### 6.11.2.5 `rsmi_dev_gpu_clk_freq_set()`

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

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

Given a device index `dv_ind`, a clock type `clk_type`, and a 64 bit bitmask `freq_bitmask`, this function will limit the set of allowable frequencies. If a bit in `freq_bitmask` has a value of 1, then the frequency (as ordered in an [rsmi\\_frequencies\\_t](#) returned by [rsmi\\_dev\\_gpu\\_clk\\_freq\\_get\(\)](#)) corresponding to that bit index will be allowed.

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

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

#### Parameters

in	<code>dv_ind</code>	a device index
in	<code>clk_type</code>	the type of clock for which the set of frequencies will be modified
in	<code>freq_bitmask</code>	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.

#### Return values

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

## 6.12 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.*
- `rsmi_status_t rsmi_dev_firmware_version_get (uint32_t dv_ind, rsmi_fw_block_t block, uint64_t *fw_version)`  
*Get the firmware versions for a device.*

### 6.12.1 Detailed Description

These functions provide version information about various subsystems.

### 6.12.2 Function Documentation

#### 6.12.2.1 `rsmi_version_get()`

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

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

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

## Parameters

<i>in, out</i>	<i>version</i>	A pointer to an <a href="#">rsmi_version_t</a> structure that will be updated with the version information upon return.
----------------	----------------	---

## Return values

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

## 6.12.2.2 rsmi\_version\_str\_get()

```
rsmi_status_t rsmi_version_str_get (
    rsmi_sw_component_t component,
    char * ver_str,
    uint32_t len )
```

Get the driver version string for the current system.

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

## Parameters

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

## Return values

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

## 6.12.2.3 rsmi\_dev\_vbios\_version\_get()

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

Get the VBIOS 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. If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.
in	<i>len</i>	The number of char's pointed to by <i>vbios</i> which can safely be written to by this function.

## Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

6.12.2.4 `rsmi_dev_firmware_version_get()`

```

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

```

Get the firmware versions for a device.

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

## Parameters

in	<i>dv_ind</i>	a device index
in	<i>block</i>	The firmware block for which the version is being requested
in, out	<i>fw_version</i>	The version for the firmware block. If this parameter is nullptr, this function will return <a href="#">RSMI_STATUS_INVALID_ARGS</a> if the function is supported with the provided arguments and <a href="#">RSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

## Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

## 6.13 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\_blocks)  
*Retrieve the enabled ECC bit-mask.*
- [rsmi\\_status\\_t rsmi\\_dev\\_ecc\\_status\\_get](#) (uint32\_t dv\_ind, [rsmi\\_gpu\\_block\\_t](#) block, [rsmi\\_ras\\_err\\_state\\_t](#) \*state)  
*Retrieve the ECC status for a GPU block.*
- [rsmi\\_status\\_t rsmi\\_status\\_string](#) ([rsmi\\_status\\_t](#) status, const char \*\*status\_string)  
*Get a description of a provided RSMI error status.*

### 6.13.1 Detailed Description

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

### 6.13.2 Function Documentation

#### 6.13.2.1 rsmi\_dev\_ecc\_count\_get()

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

Retrieve the error counts for a GPU block.

Given a device index `dv_ind`, an [rsmi\\_gpu\\_block\\_t](#) `block` and a pointer to an [rsmi\\_error\\_count\\_t](#) `ec`, this function will write the error count values for the GPU block indicated by `block` to memory pointed to by `ec`.

#### Parameters

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

#### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	call was successful
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments
<a href="#">RSMI_STATUS_INVALID_ARGS</a>	the provided arguments are not valid

### 6.13.2.2 rsmi\_dev\_ecc\_enabled\_get()

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

Retrieve the enabled ECC bit-mask.

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

#### Parameters

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

#### Return values

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

### 6.13.2.3 rsmi\_dev\_ecc\_status\_get()

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

Retrieve the ECC status for a GPU block.

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

#### Parameters

in	<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. If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function with the given arguments
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

**6.13.2.4 rsmi\_status\_string()**

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

Get a description of a provided RSMI error status.

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

## Parameters

in	<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
--	----------------------------------

**6.14 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.*

### 6.14.1 Detailed Description

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

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

The types of events available and the ability to count those events are dependent on which device is being targeted and if counters are still available for that device, respectively. [rsmi\\_dev\\_counter\\_group\\_supported\(\)](#) can be used to see which event types ([rsmi\\_event\\_group\\_t](#)) are supported for a given device. Assuming a device supports a given event type, we can then check to see if there are counters available to count a specific event with [rsmi\\_counter\\_available\\_counters\\_get\(\)](#). Counters may be occupied by other perf based programs.

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

[rsmi\\_dev\\_counter\\_create\(\)](#) allocates internal data structures that will be used to control the event counter, and return a handle to this data structure.

Once an event counter handle is obtained, the event counter can be controlled (i.e., started, stopped,...) with [rsmi\\_counter\\_control\(\)](#) by passing [rsmi\\_counter\\_command\\_t](#) commands. [RSMI\\_CNTR\\_CMD\\_START](#) starts an event counter and [RSMI\\_CNTR\\_CMD\\_STOP](#) stops a counter. [rsmi\\_counter\\_read\(\)](#) reads an event counter.

Once the counter is no longer needed, the resources it uses should be freed by calling [rsmi\\_dev\\_counter\\_destroy\(\)](#).

### 6.14.2 Important Notes about Counter Values

- A running “absolute” counter is kept internally. For the discussion that follows, we will call the internal counter value at time  $t$   $val_t$
- Issuing [RSMI\\_CNTR\\_CMD\\_START](#) or calling [rsmi\\_counter\\_read\(\)](#), causes RSMI (in kernel) to internally record the current absolute counter value
- [rsmi\\_counter\\_read\(\)](#) returns the number of events that have occurred since the previously recorded value (ie, a relative value,  $val_t - val_{t-1}$ ) from the issuing of [RSMI\\_CNTR\\_CMD\\_START](#) or calling [rsmi\\_counter\\_read\(\)](#)

Example of event counting sequence:

```

rsmi_counter_value_t value;

// Determine if RSMI_EVTNT_GRP_XGMI is supported for device dv_ind
ret = rsmi_dev_counter_group_supported(dv_ind, RSMI_EVTNT_GRP_XGMI);

// See if there are counters available for device dv_ind for event
// RSMI_EVTNT_GRP_XGMI

ret = rsmi_counter_available_counters_get(dv_ind,
                                           RSMI_EVTNT_GRP_XGMI, &counters_available);

// Assuming RSMI_EVTNT_GRP_XGMI is supported and there is at least 1
// counter available for RSMI_EVTNT_GRP_XGMI on device dv_ind, create
// an event object for an event of group RSMI_EVTNT_GRP_XGMI (e.g.,
// RSMI_EVTNT_XGMI_0_BEATS_TX) and get the handle
// (rsmi_event_handle_t).

ret = rsmi_dev_counter_create(dv_ind, RSMI_EVTNT_XGMI_0_BEATS_TX,
                              &evnt_handle);

// A program that generates the events of interest can be started
// immediately before or after starting the counters.
// Start counting:
ret = rsmi_counter_control(evnt_handle, RSMI_CNTR_CMD_START, NULL);

// Wait...

// Get the number of events since RSMI_CNTR_CMD_START was issued:
ret = rsmi_counter_read(rsmi_event_handle_t evt_handle, &value)

// Wait...

// Get the number of events since rsmi_counter_read() was last called:
ret = rsmi_counter_read(rsmi_event_handle_t evt_handle, &value)

// Stop counting.
ret = rsmi_counter_control(evnt_handle, RSMI_CNTR_CMD_STOP, NULL);

// Release all resources (e.g., counter and memory resources) associated
// with evnt_handle.
ret = rsmi_dev_counter_destroy(evnt_handle);

```

### 6.14.3 Function Documentation

#### 6.14.3.1 rsmi\_dev\_counter\_group\_supported()

```

rsmi_status_t rsmi_dev_counter_group_supported (
    uint32_t dv_ind,
    rsmi_event_group_t group )

```

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

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

##### Parameters

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

##### Return values

<a href="#">RSMI_STATUS_SUCCESS</a>	if the device associated with <code>dv_ind</code> support counting events of the type indicated by <code>group</code> .
<a href="#">RSMI_STATUS_NOT_SUPPORTED</a>	installed software or hardware does not support this function with the given arguments <code>group</code>

### 6.14.3.2 `rsmi_dev_counter_create()`

```
rsmi_status_t rsmi_dev_counter_create (
    uint32_t dv_ind,
    rsmi_event_type_t type,
    rsmi_event_handle_t * evnt_handle )
```

Create a performance counter object.

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

#### Parameters

in	<i>dv_ind</i>	a device index
in	<i>type</i>	the <code>rsmi_event_type_t</code> 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. If this parameter is nullptr, this function will return <code>RSMI_STATUS_INVALID_ARGS</code> if the function is supported with the provided arguments and <code>RSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Return values

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

### 6.14.3.3 `rsmi_dev_counter_destroy()`

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

Deallocate a performance counter object.

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

#### Parameters

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

#### Return values

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

#### 6.14.3.4 `rsmi_counter_control()`

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

Issue performance counter control commands.

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

##### Parameters

in	<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

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<a href="#"><i>RSMI_STATUS_PERMISSION</i></a>	function requires root access

#### 6.14.3.5 `rsmi_counter_read()`

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

Read the current value of a performance counter.

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

##### Parameters

in	<i>evt_handle</i>	an event handle
in, out	<i>value</i>	pointer to memory of size of <a href="#"><i>rsmi_counter_value_t</i></a> to which the counter value will be written

##### Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<a href="#"><i>RSMI_STATUS_PERMISSION</i></a>	function requires root access

#### 6.14.3.6 `rsmi_counter_available_counters_get()`

```
rsmi_status_t rsmi_counter_available_counters_get (
```

```
uint32_t dv_ind,
rsmi_event_group_t grp,
uint32_t * available )
```

Get the number of currently available counters.

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

#### Parameters

in	<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
<code>RSMI_STATUS_INVALID_ARGS</code>	the provided arguments are not valid

## 6.15 System Information Functions

### Functions

- `rsmi_status_t rsmi_compute_process_info_get (rsmi_process_info_t *procs, uint32_t *num_items)`  
*Get process information about processes currently using GPU.*
- `rsmi_status_t rsmi_compute_process_info_by_pid_get (uint32_t pid, rsmi_process_info_t *proc)`  
*Get process information about a specific process.*
- `rsmi_status_t rsmi_compute_process_gpus_get (uint32_t pid, uint32_t *dv_indices, uint32_t *num_devices)`  
*Get the device indices currently being used by a process.*

### 6.15.1 Detailed Description

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

### 6.15.2 Function Documentation

#### 6.15.2.1 `rsmi_compute_process_info_get()`

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

Get process information about processes currently using GPU.

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

## Parameters

<i>in, out</i>	<i>procs</i>	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.
<i>in, out</i>	<i>num_items</i>	A pointer to a <code>uint32_t</code> , which on input, should contain the amount of memory in <code>rsmi_process_info_t</code> '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 <code>rsmi_process_info_t</code> 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

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

6.15.2.2 `rsmi_compute_process_info_by_pid_get()`

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

Get process information about a specific process.

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

## Parameters

<i>in</i>	<i>pid</i>	The process ID for which process information is being requested
<i>in, out</i>	<i>proc</i>	a pointer to a <code>rsmi_process_info_t</code> to which process information for <code>pid</code> will be written if it is found.

## Return values

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

6.15.2.3 `rsmi_compute_process_gpus_get()`

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

Get the device indices currently being used by a process.

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

#### Parameters

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

#### Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<a href="#"><i>RSMI_STATUS_INSUFFICIENT_SIZE</i></a>	is returned if there were more gpu indices that could have been written, but not enough space was provided as indicated by <code>dv_indices</code> and <code>num_devices</code> , on input.

## 6.16 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.*
- [`rsmi\_status\_t rsmi\_dev\_xgmi\_hive\_id\_get`](#) (`uint32_t dv_ind`, `uint64_t *hive_id`)  
*Retrieve the XGMI hive id for a device.*

### 6.16.1 Detailed Description

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

## 6.16.2 Function Documentation

### 6.16.2.1 `rsmi_dev_xgmi_error_status()`

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

Retrieve the XGMI error status for a device.

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

#### Parameters

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

#### Return values

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

### 6.16.2.2 `rsmi_dev_xgmi_error_reset()`

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

Reset the XGMI error status for a device.

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

#### Parameters

in	<i>dv_ind</i>	a device index
----	---------------	----------------

#### Return values

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

### 6.16.2.3 `rsmi_dev_xgmi_hive_id_get()`

```
rsmi_status_t rsmi_dev_xgmi_hive_id_get (
```

```
uint32_t dv_ind,
uint64_t * hive_id )
```

Retrieve the XGMI hive id for a device.

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

#### Parameters

in	<code>dv_ind</code>	a device index
in, out	<code>hive_id</code>	A pointer to an <code>uint64_t</code> to which the XGMI hive id should be written

#### Return values

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

## 6.17 Hardware Topology Functions

### Functions

- `rsmi_status_t rsmi_topo_get_numa_node_number` (`uint32_t dv_ind`, `uint32_t *numa_node`)  
*Retrieve the NUMA CPU node number for a device.*
- `rsmi_status_t rsmi_topo_get_link_weight` (`uint32_t dv_ind_src`, `uint32_t dv_ind_dst`, `uint64_t *weight`)  
*Retrieve the weight for a connection between 2 GPUs.*
- `rsmi_status_t rsmi_minmax_bandwidth_get` (`uint32_t dv_ind_src`, `uint32_t dv_ind_dst`, `uint64_t *min_bandwidth`, `uint64_t *max_bandwidth`)  
*Retrieve minimal and maximal io link bandwidth between 2 GPUs.*
- `rsmi_status_t rsmi_topo_get_link_type` (`uint32_t dv_ind_src`, `uint32_t dv_ind_dst`, `uint64_t *hops`, `RSMI_IO_LINK_TYPE *type`)  
*Retrieve the hops and the connection type between 2 GPUs.*
- `rsmi_status_t rsmi_is_P2P_accessible` (`uint32_t dv_ind_src`, `uint32_t dv_ind_dst`, `bool *accessible`)  
*Return P2P availability status between 2 GPUs.*

### 6.17.1 Detailed Description

These functions are used to query Hardware topology.

## 6.17.2 Function Documentation

### 6.17.2.1 `rsmi_topo_get_numa_node_number()`

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

Retrieve the NUMA CPU node number for a device.

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

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>numa_node</i>	A pointer to an <code>uint32_t</code> to which the numa node number should be written.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

6.17.2.2 `rsmi_topo_get_link_weight()`

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

Retrieve the weight for a connection between 2 GPUs.

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

## Parameters

in	<i>dv_ind_src</i>	the source device index
in	<i>dv_ind_dst</i>	the destination device index
in, out	<i>weight</i>	A pointer to an <code>uint64_t</code> to which the weight for the connection should be written.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

6.17.2.3 `rsmi_minmax_bandwidth_get()`

```
rsmi_status_t rsmi_minmax_bandwidth_get (
    uint32_t dv_ind_src,
    uint32_t dv_ind_dst,
    uint64_t * min_bandwidth,
    uint64_t * max_bandwidth )
```

Retrieve minimal and maximal io link bandwidth between 2 GPUs.

Given a source device index `dv_ind_src` and a destination device index `dv_ind_dst`, pointer to an `uint64_t` `min_bandwidth`, and a pointer to `uint64_t` `max_bandwidth`, this function will write theoretical minimal and maximal bandwidth limits. API works if `src` and `dst` are connected via `xgmi` and have 1 hop distance.

## Parameters

in	<i>dv_ind_src</i>	the source device index
in	<i>dv_ind_dst</i>	the destination device index
in, out	<i>min_bandwidth</i>	A pointer to an <code>uint64_t</code> to which the minimal bandwidth for the connection should be written.
in, out	<i>max_bandwidth</i>	A pointer to an <code>uint64_t</code> to which the maximal bandwidth for the connection should be written.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

6.17.2.4 `rsmi_topo_get_link_type()`

```

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

```

Retrieve the hops and the connection type between 2 GPUs.

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

## Parameters

in	<i>dv_ind_src</i>	the source device index
in	<i>dv_ind_dst</i>	the destination device index
in, out	<i>hops</i>	A pointer to an <code>uint64_t</code> to which the hops for the connection should be written.
in, out	<i>type</i>	A pointer to an <a href="#"><code>RSMI_IO_LINK_TYPE</code></a> to which the type for the connection should be written.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid

6.17.2.5 `rsmi_is_P2P_accessible()`

```

rsmi_status_t rsmi_is_P2P_accessible (
    uint32_t dv_ind_src,
    uint32_t dv_ind_dst,
    bool * accessible )

```

Return P2P availability status between 2 GPUs.

Given a source device index `dv_ind_src` and a destination device index `dv_ind_dst`, and a pointer to a bool `@accessible`, this function will write the P2P connection status between the device `dv_ind_src` and `dv_ind_dst` to the memory pointed to by `accessible`.

#### Parameters

in	<code>dv_ind_src</code>	the source device index
in	<code>dv_ind_dst</code>	the destination device index
in, out	<code>accessible</code>	A pointer to a bool to which the status for the P2P connection availability should be written.

#### Return values

<a href="#"><code>RSMI_STATUS_SUCCESS</code></a>	call was successful
<a href="#"><code>RSMI_STATUS_INVALID_ARGS</code></a>	the provided arguments are not valid

## 6.18 Compute Partition Functions

### Functions

- [`rsmi\_status\_t rsmi\_dev\_compute\_partition\_get`](#) (`uint32_t dv_ind`, `char *compute_partition`, `uint32_t len`)  
*Retrieves the current compute partitioning for a desired device.*
- [`rsmi\_status\_t rsmi\_dev\_compute\_partition\_set`](#) (`uint32_t dv_ind`, [`rsmi\_compute\_partition\_type\_t`](#) `compute_partition`)  
*Modifies a selected device's compute partition setting.*
- [`rsmi\_status\_t rsmi\_dev\_compute\_partition\_reset`](#) (`uint32_t dv_ind`)  
*Reverts a selected device's compute partition setting back to its boot state.*

### 6.18.1 Detailed Description

These functions are used to configure and query the device's compute partition setting.

### 6.18.2 Function Documentation

#### 6.18.2.1 `rsmi_dev_compute_partition_get()`

```
rsmi_status_t rsmi_dev_compute_partition_get (
    uint32_t dv_ind,
    char * compute_partition,
    uint32_t len )
```

Retrieves the current compute partitioning for a desired device.

Given a device index `dv_ind` and a string `compute_partition`, and `uint32 len`, this function will attempt to obtain the device's current compute partition setting string. Upon successful retrieval, the obtained device's compute partition settings string shall be stored in the passed `compute_partition` char string variable.

## Parameters

in	<i>dv_ind</i>	a device index
in, out	<i>compute_partition</i>	a pointer to a char string variable, which the device's current compute partition will be written to.
in	<i>len</i>	the length of the caller provided buffer <i>compute_partition</i> , suggested length is 4 or greater.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<a href="#"><i>RSMI_STATUS_UNEXPECTED_DATA</i></a>	data provided to function is not valid
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function
<a href="#"><i>RSMI_STATUS_INSUFFICIENT_SIZE</i></a>	is returned if <i>len</i> bytes is not large enough to hold the entire compute partition value. In this case, only <i>len</i> bytes will be written.

6.18.2.2 `rsmi_dev_compute_partition_set()`

```
rsmi_status_t rsmi_dev_compute_partition_set (
    uint32_t dv_ind,
    rsmi_compute_partition_type_t compute_partition )
```

Modifies a selected device's compute partition setting.

Given a device index *dv\_ind*, a type of compute partition *compute\_partition*, this function will attempt to update the selected device's compute partition setting.

## Parameters

in	<i>dv_ind</i>	a device index
in	<i>compute_partition</i>	using enum <a href="#"><i>rsmi_compute_partition_type_t</i></a> , define what the selected device's compute partition setting should be updated to.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_PERMISSION</i></a>	function requires root access
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<a href="#"><i>RSMI_STATUS_SETTING_UNAVAILABLE</i></a>	the provided setting is unavailable for current device
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function

6.18.2.3 `rsmi_dev_compute_partition_reset()`

```
rsmi_status_t rsmi_dev_compute_partition_reset (
    uint32_t dv_ind )
```

Reverts a selected device's compute partition setting back to its boot state.

Given a device index `dv_ind`, this function will attempt to revert its compute partition setting back to its boot state.

## Parameters

<code>in</code>	<code>dv_ind</code>	a device index
-----------------	---------------------	----------------

## Return values

<a href="#"><code>RSMI_STATUS_SUCCESS</code></a>	call was successful
<a href="#"><code>RSMI_STATUS_PERMISSION</code></a>	function requires root access
<a href="#"><code>RSMI_STATUS_NOT_SUPPORTED</code></a>	installed software or hardware does not support this function

## 6.19 NPS Mode Functions

## Functions

- [`rsmi\_status\_t rsmi\_dev\_nps\_mode\_get`](#) (`uint32_t dv_ind`, `char *nps_mode`, `uint32_t len`)  
*Retrieves the NPS mode (memory partition) for a desired device.*
- [`rsmi\_status\_t rsmi\_dev\_nps\_mode\_set`](#) (`uint32_t dv_ind`, [`rsmi\_nps\_mode\_type\_t`](#) `nps_mode`)  
*Modifies a selected device's NPS mode (memory partition) setting.*
- [`rsmi\_status\_t rsmi\_dev\_nps\_mode\_reset`](#) (`uint32_t dv_ind`)  
*Reverts a selected device's NPS mode setting back to its boot state.*

### 6.19.1 Detailed Description

These functions are used to query the device's NPS mode (memory partition).

### 6.19.2 Function Documentation

#### 6.19.2.1 `rsmi_dev_nps_mode_get()`

```
rsmi_status_t rsmi_dev_nps_mode_get (
    uint32_t dv_ind,
    char * nps_mode,
    uint32_t len )
```

Retrieves the NPS mode (memory partition) for a desired device.

Given a device index `dv_ind` and a string `nps_mode` , and `uint32 len` , this function will attempt to obtain the device's nps mode string. Upon successful retrieval, the obtained device's nps mode string shall be stored in the passed `nps_mode` char string variable.

## Parameters

<code>in</code>	<code>dv_ind</code>	a device index
<code>in, out</code>	<code>nps_mode</code>	a pointer to a char string variable, which the device's nps mode will be written to.
<code>in</code>	<code>len</code>	the length of the caller provided buffer <code>nps_mode</code> , suggested length is 5 or greater.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<a href="#"><i>RSMI_STATUS_UNEXPECTED_DATA</i></a>	data provided to function is not valid
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function
<a href="#"><i>RSMI_STATUS_INSUFFICIENT_SIZE</i></a>	is returned if <code>len</code> bytes is not large enough to hold the entire nps mode value. In this case, only <code>len</code> bytes will be written.

6.19.2.2 `rsmi_dev_nps_mode_set()`

```
rsmi_status_t rsmi_dev_nps_mode_set (
    uint32_t dv_ind,
    rsmi_nps_mode_type_t nps_mode )
```

Modifies a selected device's NPS mode (memory partition) setting.

Given a device index `dv_ind` and a type of nps mode `nps_mode`, this function will attempt to update the selected device's nps mode setting.

## Parameters

in	<i>dv_ind</i>	a device index
in	<i>nps_mode</i>	using enum <a href="#"><i>rsmi_nps_mode_type_t</i></a> , define what the selected device's NPS mode setting should be updated to.

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	call was successful
<a href="#"><i>RSMI_STATUS_PERMISSION</i></a>	function requires root access
<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	the provided arguments are not valid
<a href="#"><i>RSMI_STATUS_NOT_SUPPORTED</i></a>	installed software or hardware does not support this function
<a href="#"><i>RSMI_STATUS_AMDGPU_RESTART_ERR</i></a>	could not successfully restart the amdgpu driver

6.19.2.3 `rsmi_dev_nps_mode_reset()`

```
rsmi_status_t rsmi_dev_nps_mode_reset (
    uint32_t dv_ind )
```

Reverts a selected device's NPS mode setting back to its boot state.

Given a device index `dv_ind`, this function will attempt to revert its NPS mode setting back to its boot state.

## Parameters

in	<i>dv_ind</i>	a device index
----	---------------	----------------

## Return values

<a href="#"><code>RSMI_STATUS_SUCCESS</code></a>	call was successful
<a href="#"><code>RSMI_STATUS_PERMISSION</code></a>	function requires root access
<a href="#"><code>RSMI_STATUS_NOT_SUPPORTED</code></a>	installed software or hardware does not support this function
<a href="#"><code>RSMI_STATUS_AMDGPU_RESTART_ERR</code></a>	could not successfully restart the amdgpu driver

## 6.20 Supported Functions

### Functions

- [`rsmi\_status\_t rsmi\_dev\_supported\_func\_iterator\_open`](#) ([`uint32\_t`](#) `dv_ind`, [`rsmi\_func\_id\_iter\_handle\_t`](#) `*handle`)  
*Get a function name iterator of supported RSMI functions for a device.*
- [`rsmi\_status\_t rsmi\_dev\_supported\_variant\_iterator\_open`](#) ([`rsmi\_func\_id\_iter\_handle\_t`](#) `obj_h`, [`rsmi\_func\_id\_iter\_handle\_t`](#) `*var_iter`)  
*Get a variant iterator for a given handle.*
- [`rsmi\_status\_t rsmi\_func\_iter\_next`](#) ([`rsmi\_func\_id\_iter\_handle\_t`](#) `handle`)  
*Advance a function identifier iterator.*
- [`rsmi\_status\_t rsmi\_dev\_supported\_func\_iterator\_close`](#) ([`rsmi\_func\_id\_iter\_handle\_t`](#) `*handle`)  
*Close a variant iterator handle.*
- [`rsmi\_status\_t rsmi\_func\_iter\_value\_get`](#) ([`rsmi\_func\_id\_iter\_handle\_t`](#) `handle`, [`rsmi\_func\_id\_value\_t`](#) `*value`)  
*Get the value associated with a function/variant iterator.*

### 6.20.1 Detailed Description

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

Some functions have several variations (“variants”) where some variants are supported and others are not. For example, on a given device, [`rsmi\_dev\_temp\_metric\_get`](#) may support some types of temperature metrics (e.g., [`RSMI\_TEMP\_CRITICAL\_HYST`](#)), but not others (e.g., [`RSMI\_TEMP\_EMERGENCY`](#)).

In addition to a top level of variant support for a function, a function may have varying support for monitors/sensors. These are considered “sub-variants” in functions described in this section. Continuing the [`rsmi\_dev\_temp\_metric\_get`](#) example, if variant [`RSMI\_TEMP\_CRITICAL\_HYST`](#) is supported, perhaps only the sub-variant sensors [`RSMI\_TEMP\_TYPE\_EDGE`](#) and [`RSMI\_TEMP\_TYPE\_EDGE`](#) are supported, but not [`RSMI\_TEMP\_TYPE\_MEMORY`](#).

In cases where a function takes in a sensor id parameter but does not have any “top level” variants, the functions in this section will indicate a default “variant”, [`RSMI\_DEFAULT\_VARIANT`](#), for the top level variant, and the various monitor support will be sub-variants of this.

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

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

```

rsmi_func_id_iter_handle_t iter_handle, var_iter, sub_var_iter;
rsmi_func_id_value_t value;
rsmi_status_t err;

for (uint32_t i = 0; i < <number of devices>; ++i) {
    std::cout << "Supported RSMI Functions:" << std::endl;
    std::cout << "\tVariants (Monitors)" << std::endl;

    err = rsmi_dev_supported_func_iterator_open(i, &iter_handle);

    while (1) {
        err = rsmi_func_iter_value_get(iter_handle, &value);
        std::cout << "Function Name: " << value.name << std::endl;

        err = rsmi_dev_supported_variant_iterator_open(iter_handle, &var_iter);
        if (err != RSMI_STATUS_NO_DATA) {
            std::cout << "\tVariants/Monitors: ";
            while (1) {
                err = rsmi_func_iter_value_get(var_iter, &value);
                if (value.id == RSMI_DEFAULT_VARIANT) {
                    std::cout << "Default Variant ";
                } else {
                    std::cout << value.id;
                }
            }
            std::cout << " (";

            err =
                rsmi_dev_supported_variant_iterator_open(var_iter, &sub_var_iter);
            if (err != RSMI_STATUS_NO_DATA) {
                while (1) {
                    err = rsmi_func_iter_value_get(sub_var_iter, &value);
                    std::cout << value.id << ", ";

                    err = rsmi_func_iter_next(sub_var_iter);

                    if (err == RSMI_STATUS_NO_DATA) {
                        break;
                    }
                }
                err = rsmi_dev_supported_func_iterator_close(&sub_var_iter);
            }

            std::cout << ")", ";";

            err = rsmi_func_iter_next(var_iter);

            if (err == RSMI_STATUS_NO_DATA) {
                break;
            }
        }
        std::cout << std::endl;

        err = rsmi_dev_supported_func_iterator_close(&var_iter);
    }

    err = rsmi_func_iter_next(iter_handle);

    if (err == RSMI_STATUS_NO_DATA) {
        break;
    }
}
err = rsmi_dev_supported_func_iterator_close(&iter_handle);
}

```

## 6.20.2 Function Documentation

### 6.20.2.1 rsmi\_dev\_supported\_func\_iterator\_open()

```

rsmi_status_t rsmi_dev_supported_func_iterator_open (
    uint32_t dv_ind,
    rsmi_func_id_iter_handle_t * handle )

```

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

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

Note that although this function takes in `dv_ind` as an argument, [rsmi\\_dev\\_supported\\_func\\_iterator\\_open](#) itself will not be among the functions listed as supported. This is because [rsmi\\_dev\\_supported\\_func\\_iterator\\_open](#) does not depend on hardware or driver support and should always be supported.

## Parameters

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

## Return values

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

6.20.2.2 `rsmi_dev_supported_variant_iterator_open()`

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

Get a variant iterator for a given handle.

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

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

## Parameters

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

## Return values

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

6.20.2.3 `rsmi_func_iter_next()`

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

Advance a function identifier iterator.

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

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

**Parameters**

in	<i>handle</i>	A pointer to an iterator handle to be incremented
----	---------------	---

**Return values**

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call.
<a href="#"><i>RSMI_STATUS_NO_DATA</i></a>	is returned when list of identifiers has been exhausted

**6.20.2.4 rsmi\_dev\_supported\_func\_iterator\_close()**

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

Close a variant iterator handle.

Given a pointer to an [\*rsmi\\_func\\_id\\_iter\\_handle\\_t\*](#) handle, this function will free the resources being used by the handle

**Parameters**

in	<i>handle</i>	A pointer to an iterator handle to be closed
----	---------------	--

**Return values**

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

**6.20.2.5 rsmi\_func\_iter\_value\_get()**

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

Get the value associated with a function/variant iterator.

Given an [\*rsmi\\_func\\_id\\_iter\\_handle\\_t\*](#) handle, this function will write the identifier of the function/variant to the user provided memory pointed to by *value*.

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

**Parameters**

in	<i>handle</i>	An iterator for which the value is being requested
in, out	<i>value</i>	A pointer to an <a href="#"><i>rsmi_func_id_value_t</i></a> provided by the caller to which this function will write the value associated with <i>handle</i>

## Return values

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

## 6.21 Event Notification Functions

### Functions

- `rsmi_status_t rsmi_event_notification_init` (uint32\_t dv\_ind)  
*Prepare to collect event notifications for a GPU.*
- `rsmi_status_t rsmi_event_notification_mask_set` (uint32\_t dv\_ind, uint64\_t mask)  
*Specify which events to collect for a device.*
- `rsmi_status_t rsmi_event_notification_get` (int timeout\_ms, uint32\_t \*num\_elem, `rsmi_evt_notification_data_t` \*data)  
*Collect event notifications, waiting a specified amount of time.*
- `rsmi_status_t rsmi_event_notification_stop` (uint32\_t dv\_ind)  
*Close any file handles and free any resources used by event notification for a GPU.*

### 6.21.1 Detailed Description

These functions are used to configure for and get asynchronous event notifications.

### 6.21.2 Function Documentation

#### 6.21.2.1 `rsmi_event_notification_init()`

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

Prepare to collect event notifications for a GPU.

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

## Parameters

<code>dv_ind</code>	a device index corresponding to the device on which to listen for events
---------------------	--

## Return values

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

#### 6.21.2.2 `rsmi_event_notification_mask_set()`

```
rsmi_status_t rsmi_event_notification_mask_set (
```

```
uint32_t dv_ind,
uint64_t mask )
```

Specify which events to collect for a device.

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

#### Parameters

<i>dv_ind</i>	a device index corresponding to the device on which to listen for events
<i>mask</i>	Bitmask generated by OR'ing 1 or more elements of <code>rsmi_evt_notification_type_t</code> indicating which event types to listen for, where the <code>rsmi_evt_notification_type_t</code> value indicates the bit field, with bit position starting from 1. For example, if the mask field is 0x0000000000000003, which means first bit, bit 1 (bit position start from 1) and bit 2 are set, which indicate interest in receiving <code>RSMI_EVT_NOTIF_VMFault</code> (which has a value of 1) and <code>RSMI_EVT_NOTIF_THERMAL_THROTTLE</code> event (which has a value of 2).

#### Return values

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

### 6.21.2.3 rsmi\_event\_notification\_get()

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

Collect event notifications, waiting a specified amount of time.

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

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

#### Parameters

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

## Return values

<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	The function ran successfully. The events that were found are written to <code>data</code> and <code>num_elems</code> is updated with the number of elements that were written.
<a href="#"><i>RSMI_STATUS_NO_DATA</i></a>	No events were found to collect.

6.21.2.4 `rsmi_event_notification_stop()`

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

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

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

## Parameters

in	<code>dv_ind</code>	The device index of the GPU for which event notification resources will be free
----	---------------------	---

## Return values

<a href="#"><i>RSMI_STATUS_INVALID_ARGS</i></a>	resources for the given device have either already been freed, or were never allocated by <a href="#"><code>rsmi_event_notification_init()</code></a>
<a href="#"><i>RSMI_STATUS_SUCCESS</i></a>	is returned upon successful call



# Chapter 7

## Data Structure Documentation

### 7.1 id Union Reference

This union holds the value of an [rsmi\\_func\\_id\\_iter\\_handle\\_t](#). The value may be a function name, or an enumerated variant value of types such as [rsmi\\_memory\\_type\\_t](#), [rsmi\\_temperature\\_metric\\_t](#), etc.

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

#### Data Fields

- `uint64_t id`  
*uint64\_t representation of value*
  - `const char * name`  
*name string (applicable to functions only)*
  - union {
    - [rsmi\\_memory\\_type\\_t](#) `memory_type`  
*< Used for [rsmi\\_memory\\_type\\_t](#) variants*
    - [rsmi\\_temperature\\_metric\\_t](#) `temp_metric`  
*Used for [rsmi\\_event\\_type\\_t](#) variants.*
    - [rsmi\\_event\\_type\\_t](#) `evnt_type`  
*Used for [rsmi\\_event\\_group\\_t](#) variants.*
    - [rsmi\\_event\\_group\\_t](#) `evnt_group`  
*Used for [rsmi\\_clk\\_type\\_t](#) variants.*
    - [rsmi\\_clk\\_type\\_t](#) `clk_type`  
*Used for [rsmi\\_fw\\_block\\_t](#) variants.*
    - [rsmi\\_fw\\_block\\_t](#) `fw_block`  
*Used for [rsmi\\_gpu\\_block\\_t](#) variants.*
    - [rsmi\\_gpu\\_block\\_t](#) `gpu_block_type`
- ```
};
```

#### 7.1.1 Detailed Description

This union holds the value of an [rsmi\\_func\\_id\\_iter\\_handle\\_t](#). The value may be a function name, or an enumerated variant value of types such as [rsmi\\_memory\\_type\\_t](#), [rsmi\\_temperature\\_metric\\_t](#), etc.

## 7.1.2 Field Documentation

### 7.1.2.1 memory\_type

`rsmi_memory_type_t id::memory_type`

< Used for `rsmi_memory_type_t` variants

Used for `rsmi_temperature_metric_t` variants

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

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

## 7.2 metrics\_table\_header\_t Struct Reference

The following structures hold the gpu metrics values for a device.

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

### 7.2.1 Detailed Description

The following structures hold the gpu metrics values for a device.

Size and version information of metrics data

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

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

## 7.3 rsmi\_counter\_value\_t Struct Reference

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

### Data Fields

- `uint64_t value`  
*Counter value.*
- `uint64_t time_enabled`
- `uint64_t time_running`

### 7.3.1 Detailed Description

Counter value

### 7.3.2 Field Documentation

#### 7.3.2.1 time\_enabled

```
uint64_t rsmi_counter_value_t::time_enabled
```

Time that the counter was enabled (in nanoseconds)

#### 7.3.2.2 time\_running

```
uint64_t rsmi_counter_value_t::time_running
```

Time that the counter was running (in nanoseconds)

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

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

## 7.4 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.*

### 7.4.1 Detailed Description

This structure holds error counts.

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

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

## 7.5 rsmi\_evt\_notification\_data\_t Struct Reference

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

## Data Fields

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

### 7.5.1 Detailed Description

Event notification data returned from event notification API

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

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

## 7.6 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.*

### 7.6.1 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](#)

## 7.7 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](#)]

## 7.7.1 Detailed Description

This structure holds information about clock frequencies.

## 7.7.2 Field Documentation

### 7.7.2.1 num\_supported

```
uint32_t rsmi_frequencies_t::num_supported
```

The number of supported frequencies

### 7.7.2.2 current

```
uint32_t rsmi_frequencies_t::current
```

The current frequency index

### 7.7.2.3 frequency

```
uint64_t rsmi_frequencies_t::frequency [RSMI\_MAX\_NUM\_FREQUENCIES]
```

List of frequencies. Only the first num\_supported frequencies are valid.

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

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

## 7.8 rsmi\_gpu\_metrics\_t Struct Reference

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

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

## 7.9 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)*

### 7.9.1 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](#)

## 7.10 `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`]

### 7.10.1 Detailed Description

[RSMI\\_NUM\\_VOLTAGE\\_CURVE\\_POINTS](#) number of [rsmi\\_od\\_vddc\\_point\\_t](#)'s

### 7.10.2 Field Documentation

#### 7.10.2.1 `vc_points`

```
rsmi\_od\_vddc\_point\_t rsmi_od_volt_curve_t::vc_points [RSMI_NUM_VOLTAGE_CURVE_POINTS]
```

Array of [RSMI\\_NUM\\_VOLTAGE\\_CURVE\\_POINTS](#) [rsmi\\_od\\_vddc\\_point\\_t](#)'s that make up the voltage frequency curve points.

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

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

## 7.11 `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.*

**7.11.1 Detailed Description**

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

**7.11.2 Field Documentation****7.11.2.1 curr\_mclk\_range**

```
rsmi\_range\_t rsmi_od_volt_freq_data_t::curr_mclk_range
```

The current MCLK frequency range; (upper bound only)

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

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

**7.12 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](#)]

**7.12.1 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.

## 7.12.2 Field Documentation

### 7.12.2.1 transfer\_rate

`rsmi_frequencies_t rsmi_pcie_bandwidth_t::transfer_rate`

Transfer rates (T/s) that are possible

### 7.12.2.2 lanes

`uint32_t rsmi_pcie_bandwidth_t::lanes[RSMI_MAX_NUM_FREQUENCIES]`

List of lanes for corresponding transfer rate. Only the first num\_supported bandwidths are valid.

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

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

## 7.13 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`

### 7.13.1 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.

## 7.13.2 Field Documentation

### 7.13.2.1 available\_profiles

`rsmi_bit_field_t rsmi_power_profile_status_t::available_profiles`

Which profiles are supported by this system

### 7.13.2.2 current

```
rsmi_power_profile_preset_masks_t rsmi_power_profile_status_t::current
```

Which power profile is currently active

### 7.13.2.3 num\_profiles

```
uint32_t rsmi_power_profile_status_t::num_profiles
```

How many power profiles are available

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

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

## 7.14 rsmi\_process\_info\_t Struct Reference

This structure contains information specific to a process.

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

### Data Fields

- `uint32_t process_id`  
*Process ID.*
- `uint32_t pasid`  
*PASID.*
- `uint64_t vram_usage`  
*VRAM usage.*
- `uint64_t sdma_usage`  
*SDMA usage in microseconds.*
- `uint32_t cu_occupancy`  
*Compute Unit usage in percent.*

### 7.14.1 Detailed Description

This structure contains information specific to a process.

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

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

## 7.15 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.*

### 7.15.1 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](#)

## 7.16 rsmi\_retired\_page\_record\_t Struct Reference

Reserved Memory Page Record.

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

### Data Fields

- **uint64\_t page\_address**  
*Start address of page.*
- **uint64\_t page\_size**  
*Page size.*
- [rsmi\\_memory\\_page\\_status\\_t](#) **status**  
*Page "reserved" status.*

### 7.16.1 Detailed Description

Reserved Memory Page Record.

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

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

## 7.17 rsmi\_utilization\_counter\_t Struct Reference

The utilization counter data.

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

### Data Fields

- [RSMI\\_UTILIZATION\\_COUNTER\\_TYPE](#) **type**  
*Utilization counter type.*
- `uint64_t` **value**  
*Utilization counter value.*

### 7.17.1 Detailed Description

The utilization counter data.

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

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

## 7.18 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.*

### 7.18.1 Detailed Description

This structure holds version information.

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

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



# Chapter 8

## File Documentation

### 8.1 rocm\_smi.h File Reference

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

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

#### Data Structures

- struct [rsmi\\_counter\\_value\\_t](#)
- struct [rsmi\\_evt\\_notification\\_data\\_t](#)
- struct [rsmi\\_utilization\\_counter\\_t](#)  
*The utilization counter data.*
- struct [rsmi\\_retired\\_page\\_record\\_t](#)  
*Reserved Memory Page Record.*
- struct [rsmi\\_power\\_profile\\_status\\_t](#)  
*This structure contains information about which power profiles are supported by the system for a given device, and which power profile is currently active.*
- struct [rsmi\\_frequencies\\_t](#)  
*This structure holds information about clock frequencies.*
- struct [rsmi\\_pcie\\_bandwidth\\_t](#)  
*This structure holds information about the possible PCIe bandwidths. Specifically, the possible transfer rates and their associated numbers of lanes are stored here.*
- struct [rsmi\\_version\\_t](#)  
*This structure holds version information.*
- struct [rsmi\\_range\\_t](#)  
*This structure represents a range (e.g., frequencies or voltages).*
- struct [rsmi\\_od\\_vddc\\_point\\_t](#)  
*This structure represents a point on the frequency-voltage plane.*
- struct [rsmi\\_freq\\_volt\\_region\\_t](#)

This structure holds 2 [rsmi\\_range\\_t](#)'s, one for frequency and one for voltage. These 2 ranges indicate the range of possible values for the corresponding [rsmi\\_od\\_vddc\\_point\\_t](#).

- struct [rsmi\\_od\\_volt\\_curve\\_t](#)
- struct [rsmi\\_od\\_volt\\_freq\\_data\\_t](#)

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

- struct [metrics\\_table\\_header\\_t](#)

The following structures hold the gpu metrics values for a device.

- struct [rsmi\\_gpu\\_metrics\\_t](#)
- struct [rsmi\\_error\\_count\\_t](#)

This structure holds error counts.

- struct [rsmi\\_process\\_info\\_t](#)

This structure contains information specific to a process.

- union [id](#)

This union holds the value of an [rsmi\\_func\\_id\\_iter\\_handle\\_t](#). The value may be a function name, or an enumerated variant value of types such as [rsmi\\_memory\\_type\\_t](#), [rsmi\\_temperature\\_metric\\_t](#), etc.

## Macros

- #define **RSMI\_MAX\_NUM\_FREQUENCIES** 32

Guaranteed maximum possible number of supported frequencies.

- #define [RSMI\\_MAX\\_FAN\\_SPEED](#) 255
- #define **RSMI\_NUM\_VOLTAGE\_CURVE\_POINTS** 3

The number of points that make up a voltage-frequency curve definition.

- #define [RSMI\\_EVENT\\_MASK\\_FROM\\_INDEX](#)(i) (1ULL << ((i) - 1))
- #define **MAX\_EVENT\_NOTIFICATION\_MSG\_SIZE** 64

Maximum number of characters an event notification message will be.

- #define **RSMI\_MAX\_NUM\_POWER\_PROFILES** (sizeof([rsmi\\_bit\\_field\\_t](#)) \* 8)

Number of possible power profiles that a system could support.

- #define **RSMI\_GPU\_METRICS\_API\_FORMAT\_VER** 1

The following structure holds the gpu metrics values for a device.

- #define **RSMI\_GPU\_METRICS\_API\_CONTENT\_VER\_1** 1
- #define **RSMI\_GPU\_METRICS\_API\_CONTENT\_VER\_2** 2
- #define **RSMI\_GPU\_METRICS\_API\_CONTENT\_VER\_3** 3
- #define **RSMI\_NUM\_HBM\_INSTANCES** 4
- #define **CENTRIGRADE\_TO\_MILLI\_CENTIGRADE** 1000
- #define [RSMI\\_DEFAULT\\_VARIANT](#) 0xFFFFFFFFFFFFFFFF

## Typedefs

- typedef uintptr\_t [rsmi\\_event\\_handle\\_t](#)

Handle to performance event counter.

- typedef uint64\_t **rsmi\_bit\_field\_t**

Bitfield used in various RSMI calls.

- typedef enum [\\_RSMI\\_IO\\_LINK\\_TYPE](#) **RSMI\_IO\_LINK\_TYPE**

Types for IO Link.

- typedef struct rsmi\_func\_id\_iter\_handle \* **rsmi\_func\_id\_iter\_handle\_t**

Opaque handle to function-support object.

- typedef union [id](#) **rsmi\_func\_id\_value\_t**

This union holds the value of an [rsmi\\_func\\_id\\_iter\\_handle\\_t](#). The value may be a function name, or an enumerated variant value of types such as [rsmi\\_memory\\_type\\_t](#), [rsmi\\_temperature\\_metric\\_t](#), etc.

## Enumerations

- enum `rsmi_status_t` {  
`RSMI_STATUS_SUCCESS` = 0x0 , `RSMI_STATUS_INVALID_ARGS` , `RSMI_STATUS_NOT_SUPPORTED` ,  
`RSMI_STATUS_FILE_ERROR` ,  
`RSMI_STATUS_PERMISSION` , `RSMI_STATUS_OUT_OF_RESOURCES` , `RSMI_STATUS_INTERNAL_EXCEPTION` ,  
`RSMI_STATUS_INPUT_OUT_OF_BOUNDS` ,  
`RSMI_STATUS_INIT_ERROR` , **`RSMI_INITIALIZATION_ERROR`** = `RSMI_STATUS_INIT_ERROR` ,  
`RSMI_STATUS_NOT_YET_IMPLEMENTED` , `RSMI_STATUS_NOT_FOUND` ,  
`RSMI_STATUS_INSUFFICIENT_SIZE` , `RSMI_STATUS_INTERRUPT` , `RSMI_STATUS_UNEXPECTED_SIZE` ,  
`RSMI_STATUS_NO_DATA` ,  
`RSMI_STATUS_UNEXPECTED_DATA` , `RSMI_STATUS_BUSY` , `RSMI_STATUS_REFCOUNT_OVERFLOW` ,  
`RSMI_STATUS_SETTING_UNAVAILABLE` ,  
`RSMI_STATUS_AMDGPU_RESTART_ERR` , `RSMI_STATUS_UNKNOWN_ERROR` = 0xFFFFFFFF }  
*Error codes returned by rocm\_smi\_lib functions.*
- enum `rsmi_init_flags_t` { `RSMI_INIT_FLAG_ALL_GPUS` = 0x1 , `RSMI_INIT_FLAG_RESRV_TEST1` = 0x8000000000000000 }  
*Initialization flags.*
- enum `rsmi_dev_perf_level_t` {  
`RSMI_DEV_PERF_LEVEL_AUTO` = 0 , **`RSMI_DEV_PERF_LEVEL_FIRST`** = `RSMI_DEV_PERF_LEVEL_AUTO` , `RSMI_DEV_PERF_LEVEL_LOW` , `RSMI_DEV_PERF_LEVEL_HIGH` ,  
`RSMI_DEV_PERF_LEVEL_MANUAL` , `RSMI_DEV_PERF_LEVEL_STABLE_STD` , `RSMI_DEV_PERF_LEVEL_STABLE_PEA` ,  
`RSMI_DEV_PERF_LEVEL_STABLE_MIN_MCLK` ,  
`RSMI_DEV_PERF_LEVEL_STABLE_MIN_SCLK` , `RSMI_DEV_PERF_LEVEL_DETERMINISM` , **`RSMI_DEV_PERF_LEVEL_LAST`** = `RSMI_DEV_PERF_LEVEL_DETERMINISM` , `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_XGMI_DATA_OUT` = 10 , **`RSMI_EVNT_GRP_INVALID`** = 0xFFFFFFFF }  
*Enum denoting an event group. The value of the enum is the base value for all the event enums in the group.*
- enum `rsmi_event_type_t` {  
**`RSMI_EVNT_FIRST`** = `RSMI_EVNT_GRP_XGMI` , **`RSMI_EVNT_XGMI_FIRST`** = `RSMI_EVNT_GRP_XGMI` ,  
`RSMI_EVNT_XGMI_0_NOP_TX` = `RSMI_EVNT_XGMI_FIRST` , `RSMI_EVNT_XGMI_0_REQUEST_TX` ,  
`RSMI_EVNT_XGMI_0_RESPONSE_TX` , `RSMI_EVNT_XGMI_0_BEATS_TX` , `RSMI_EVNT_XGMI_1_NOP_TX` ,  
`RSMI_EVNT_XGMI_1_REQUEST_TX` ,  
`RSMI_EVNT_XGMI_1_RESPONSE_TX` , `RSMI_EVNT_XGMI_1_BEATS_TX` , **`RSMI_EVNT_XGMI_LAST`**  
= `RSMI_EVNT_XGMI_1_BEATS_TX` , **`RSMI_EVNT_XGMI_DATA_OUT_FIRST`** = `RSMI_EVNT_GRP_XGMI_DATA_OUT` ,  
**`RSMI_EVNT_XGMI_DATA_OUT_0`** = `RSMI_EVNT_XGMI_DATA_OUT_FIRST` , `RSMI_EVNT_XGMI_DATA_OUT_1` ,  
`RSMI_EVNT_XGMI_DATA_OUT_2` , `RSMI_EVNT_XGMI_DATA_OUT_3` ,  
`RSMI_EVNT_XGMI_DATA_OUT_4` , `RSMI_EVNT_XGMI_DATA_OUT_5` , **`RSMI_EVNT_XGMI_DATA_OUT_LAST`** = `RSMI_EVNT_XGMI_DATA_OUT_5` , **`RSMI_EVNT_LAST`** = `RSMI_EVNT_XGMI_DATA_OUT_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_evt_notification_type_t` {  
`RSMI_EVT_NOTIF_VMFault` = `KFD_SMI_EVENT_VMFault` , **`RSMI_EVT_NOTIF_FIRST`** = `RSMI_EVT_NOTIF_VMFault` , **`RSMI_EVT_NOTIF_THERMAL_THROTTLE`** = `KFD_SMI_EVENT_THERMAL_THROTTLE` ,  
`RSMI_EVT_NOTIF_GPU_PRE_RESET` = `KFD_SMI_EVENT_GPU_PRE_RESET` ,  
`RSMI_EVT_NOTIF_GPU_POST_RESET` = `KFD_SMI_EVENT_GPU_POST_RESET` , **`RSMI_EVT_NOTIF_LAST`** = `RSMI_EVT_NOTIF_GPU_POST_RESET` }

- 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_PCIE` , `RSMI_CLK_TYPE_LAST` =  
`RSMI_CLK_TYPE_MEM` ,  
`RSMI_CLK_INVALID` = 0xFFFFFFFF }
- enum `rsmi_compute_partition_type_t` {  
`RSMI_COMPUTE_PARTITION_INVALID` = 0 , `RSMI_COMPUTE_PARTITION_CPX` , `RSMI_COMPUTE_PARTITION_SPX` ,  
`RSMI_COMPUTE_PARTITION_DPX` ,  
`RSMI_COMPUTE_PARTITION_TPX` , `RSMI_COMPUTE_PARTITION_QPX` }  

*Compute Partition. This enum is used to identify various compute partitioning settings.*
- enum `rsmi_nps_mode_type_t` {  
`RSMI_MEMORY_PARTITION_UNKNOWN` = 0 , `RSMI_MEMORY_PARTITION_NPS1` , `RSMI_MEMORY_PARTITION_NPS2` ,  
`RSMI_MEMORY_PARTITION_NPS4` ,  
`RSMI_MEMORY_PARTITION_NPS8` }  

*NPS Modes. This enum is used to identify various NPS mode types.*
- enum `rsmi_temperature_metric_t` {  
`RSMI_TEMP_CURRENT` = 0x0 , `RSMI_TEMP_FIRST` = `RSMI_TEMP_CURRENT` , `RSMI_TEMP_MAX` ,  
`RSMI_TEMP_MIN` ,  
`RSMI_TEMP_MAX_HYST` , `RSMI_TEMP_MIN_HYST` , `RSMI_TEMP_CRITICAL` , `RSMI_TEMP_CRITICAL_HYST` ,  
`RSMI_TEMP_EMERGENCY` , `RSMI_TEMP_EMERGENCY_HYST` , `RSMI_TEMP_CRIT_MIN` , `RSMI_TEMP_CRIT_MIN_HYST` ,  
`RSMI_TEMP_OFFSET` , `RSMI_TEMP_LOWEST` , `RSMI_TEMP_HIGHEST` , `RSMI_TEMP_LAST` = `RSMI_TEMP_HIGHEST` }  

*Temperature Metrics. This enum is used to identify various temperature metrics. Corresponding values will be in 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_HBM_0` , `RSMI_TEMP_TYPE_HBM_1` , `RSMI_TEMP_TYPE_HBM_2` , `RSMI_TEMP_TYPE_HBM_3` ,  
`RSMI_TEMP_TYPE_LAST` = `RSMI_TEMP_TYPE_HBM_3` , `RSMI_TEMP_TYPE_INVALID` = 0xFFFFFFFF }  

*This enumeration is used to indicate from which part of the device a temperature reading should be obtained.*
- enum `rsmi_voltage_metric_t` {  
`RSMI_VOLT_CURRENT` = 0x0 , `RSMI_VOLT_FIRST` = `RSMI_VOLT_CURRENT` , `RSMI_VOLT_MAX` ,  
`RSMI_VOLT_MIN_CRIT` ,  
`RSMI_VOLT_MIN` , `RSMI_VOLT_MAX_CRIT` , `RSMI_VOLT_AVERAGE` , `RSMI_VOLT_LOWEST` ,  
`RSMI_VOLT_HIGHEST` , `RSMI_VOLT_LAST` = `RSMI_VOLT_HIGHEST` }  

*Voltage Metrics. This enum is used to identify various Voltage metrics. Corresponding values will be in millivolt.*
- enum `rsmi_voltage_type_t` { `RSMI_VOLT_TYPE_FIRST` = 0 , `RSMI_VOLT_TYPE_VDDGFX` = `RSMI_VOLT_TYPE_FIRST` , `RSMI_VOLT_TYPE_LAST` = `RSMI_VOLT_TYPE_VDDGFX` , `RSMI_VOLT_TYPE_INVALID` = 0xFFFFFFFF }  

*This enumeration is used to indicate which type of voltage reading should be obtained.*
- enum `rsmi_power_profile_preset_masks_t` {  
`RSMI_PWR_PROF_PRST_CUSTOM_MASK` = 0x1 , `RSMI_PWR_PROF_PRST_VIDEO_MASK` = 0x2 ,  
`RSMI_PWR_PROF_PRST_POWER_SAVING_MASK` = 0x4 , `RSMI_PWR_PROF_PRST_COMPUTE_MASK` = 0x8 ,  
`RSMI_PWR_PROF_PRST_VR_MASK` = 0x10 , `RSMI_PWR_PROF_PRST_3D_FULL_SCR_MASK` = 0x20 ,  
`RSMI_PWR_PROF_PRST_BOOTUP_DEFAULT` = 0x40 , `RSMI_PWR_PROF_PRST_LAST` = `RSMI_PWR_PROF_PRST_BOOTUP_DEFAULT` ,  
`RSMI_PWR_PROF_PRST_INVALID` = 0xFFFFFFFFFFFFFFFF }  

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

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

*This enum is used to identify different GPU blocks.*

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

*The current ECC state.*

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

*Types of memory.*

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

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

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

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

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

*XGMI Status.*

- enum `rsmi_memory_page_status_t` { `RSMI_MEM_PAGE_STATUS_RESERVED` = 0 , `RSMI_MEM_PAGE_STATUS_PENDING`  
`RSMI_MEM_PAGE_STATUS_UNRESERVABLE` }

*Reserved Memory Page States.*

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

*Types for IO Link.*

- enum `RSMI_UTILIZATION_COUNTER_TYPE` { `RSMI_UTILIZATION_COUNTER_FIRST` = 0 , `RSMI_COARSE_GRAIN_GFX_ACTIVITY` = `RSMI_UTILIZATION_COUNTER_FIRST` , `RSMI_COARSE_GRAIN_MEM_ACTIVITY`  
`RSMI_UTILIZATION_COUNTER_LAST` = `RSMI_COARSE_GRAIN_MEM_ACTIVITY` }

The utilization counter type.

## 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\\_sku\\_get](#) (uint32\_t dv\_ind, char \*sku)  
*Get the SKU for a desired device associated with the device with provided device index.*
- [rsmi\\_status\\_t rsmi\\_dev\\_vendor\\_id\\_get](#) (uint32\_t dv\_ind, uint16\_t \*id)  
*Get the device vendor id associated with the device with provided device index.*
- [rsmi\\_status\\_t rsmi\\_dev\\_name\\_get](#) (uint32\_t dv\_ind, char \*name, size\_t len)  
*Get the name string of a gpu device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_brand\\_get](#) (uint32\_t dv\_ind, char \*brand, uint32\_t len)  
*Get the brand string of a gpu device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_vendor\\_name\\_get](#) (uint32\_t dv\_ind, char \*name, size\_t len)  
*Get the name string for a give vendor ID.*
- [rsmi\\_status\\_t rsmi\\_dev\\_vram\\_vendor\\_get](#) (uint32\_t dv\_ind, char \*brand, uint32\_t len)  
*Get the vram vendor string of a gpu device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_serial\\_number\\_get](#) (uint32\_t dv\_ind, char \*serial\_num, uint32\_t len)  
*Get the serial number string for a device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_subsystem\\_id\\_get](#) (uint32\_t dv\_ind, uint16\_t \*id)  
*Get the subsystem device id associated with the device with provided device index.*
- [rsmi\\_status\\_t rsmi\\_dev\\_subsystem\\_name\\_get](#) (uint32\_t dv\_ind, char \*name, size\_t len)  
*Get the name string for the device subsystem.*
- [rsmi\\_status\\_t rsmi\\_dev\\_drm\\_render\\_minor\\_get](#) (uint32\_t dv\_ind, uint32\_t \*minor)  
*Get the drm minor number associated with this device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_subsystem\\_vendor\\_id\\_get](#) (uint32\_t dv\_ind, uint16\_t \*id)  
*Get the device subsystem vendor id associated with the device with provided device index.*
- [rsmi\\_status\\_t rsmi\\_dev\\_unique\\_id\\_get](#) (uint32\_t dv\_ind, uint64\_t \*id)  
*Get Unique ID.*
- [rsmi\\_status\\_t rsmi\\_dev\\_pcie\\_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\\_pcie\\_id\\_get](#) (uint32\_t dv\_ind, uint64\_t \*bdfid)  
*Get the unique PCI device identifier associated for a device.*
- [rsmi\\_status\\_t rsmi\\_topo\\_numa\\_affinity\\_get](#) (uint32\_t dv\_ind, uint32\_t \*numa\_node)  
*Get the NUMA node associated with a device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_pcie\\_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\\_pcie\\_replay\\_counter\\_get](#) (uint32\_t dv\_ind, uint64\_t \*counter)  
*Get PCIe replay counter.*
- [rsmi\\_status\\_t rsmi\\_dev\\_pcie\\_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\\_energy\\_count\\_get](#) (uint32\_t dv\_ind, uint64\_t \*power, float \*counter\_resolution, uint64\_t \*timestamp)

*Get the energy accumulator counter 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\\_default\\_get](#) (uint32\_t dv\_ind, uint64\_t \*default\_cap)

*Get the default power cap for the device specified by dv\_ind.*

- [rsmi\\_status\\_t rsmi\\_dev\\_power\\_cap\\_range\\_get](#) (uint32\_t dv\_ind, uint32\_t sensor\_ind, uint64\_t \*max, uint64\_t \*min)

*Get the range of valid values for the power cap.*

- [rsmi\\_status\\_t rsmi\\_dev\\_power\\_cap\\_set](#) (uint32\_t dv\_ind, uint32\_t sensor\_ind, uint64\_t cap)

*Set the power cap value.*

- [rsmi\\_status\\_t rsmi\\_dev\\_power\\_profile\\_set](#) (uint32\_t dv\_ind, uint32\_t reserved, [rsmi\\_power\\_profile\\_preset\\_masks\\_t](#) profile)

*Set the power profile.*

- [rsmi\\_status\\_t rsmi\\_dev\\_memory\\_total\\_get](#) (uint32\_t dv\_ind, [rsmi\\_memory\\_type\\_t](#) mem\_type, uint64\_t \*total)

*Get the total amount of memory that exists.*

- [rsmi\\_status\\_t rsmi\\_dev\\_memory\\_usage\\_get](#) (uint32\_t dv\_ind, [rsmi\\_memory\\_type\\_t](#) mem\_type, uint64\_t \*used)

*Get the current memory usage.*

- [rsmi\\_status\\_t rsmi\\_dev\\_memory\\_busy\\_percent\\_get](#) (uint32\_t dv\_ind, uint32\_t \*busy\_percent)

*Get percentage of time any device memory is being used.*

- [rsmi\\_status\\_t rsmi\\_dev\\_memory\\_reserved\\_pages\\_get](#) (uint32\_t dv\_ind, uint32\_t \*num\_pages, [rsmi\\_retired\\_page\\_record\\_t](#) \*records)

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

- [rsmi\\_status\\_t rsmi\\_dev\\_fan\\_rpms\\_get](#) (uint32\_t dv\_ind, uint32\_t sensor\_ind, int64\_t \*speed)

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

- [rsmi\\_status\\_t rsmi\\_dev\\_fan\\_speed\\_get](#) (uint32\_t dv\_ind, uint32\_t sensor\_ind, int64\_t \*speed)

*Get the fan speed for the specified device as a value relative to [RSMI\\_MAX\\_FAN\\_SPEED](#).*

- [rsmi\\_status\\_t rsmi\\_dev\\_fan\\_speed\\_max\\_get](#) (uint32\_t dv\_ind, uint32\_t sensor\_ind, uint64\_t \*max\_speed)

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

- [rsmi\\_status\\_t rsmi\\_dev\\_temp\\_metric\\_get](#) (uint32\_t dv\_ind, uint32\_t sensor\_type, [rsmi\\_temperature\\_metric\\_t](#) metric, int64\_t \*temperature)

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

- [rsmi\\_status\\_t rsmi\\_dev\\_volt\\_metric\\_get](#) (uint32\_t dv\_ind, [rsmi\\_voltage\\_type\\_t](#) sensor\_type, [rsmi\\_voltage\\_metric\\_t](#) metric, int64\_t \*voltage)

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

- [rsmi\\_status\\_t rsmi\\_dev\\_fan\\_reset](#) (uint32\_t dv\_ind, uint32\_t sensor\_ind)

*Reset the fan to automatic driver control.*

- [rsmi\\_status\\_t rsmi\\_dev\\_fan\\_speed\\_set](#) (uint32\_t dv\_ind, uint32\_t sensor\_ind, uint64\_t speed)

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

- [rsmi\\_status\\_t rsmi\\_dev\\_busy\\_percent\\_get](#) (uint32\_t dv\_ind, uint32\_t \*busy\_percent)

*Get percentage of time device is busy doing any processing.*

- [rsmi\\_status\\_t rsmi\\_utilization\\_count\\_get](#) (uint32\_t dv\_ind, [rsmi\\_utilization\\_counter\\_t](#) utilization\_counters[], uint32\_t count, uint64\_t \*timestamp)

*Get coarse grain utilization counter of the specified device.*

- [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\\_perf\\_determinism\\_mode\\_set](#) (uint32\_t dv\_ind, uint64\_t clkvalue)

- Enter performance determinism mode 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\\_mem\\_overdrive\\_level\\_get](#) (uint32\_t dv\_ind, uint32\_t \*od)

*Get the memory clock 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\\_gpu\\_reset](#) (int32\_t dv\_ind)

*Reset the gpu associated with the device with provided device index.*
- [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\\_gpu\\_metrics\\_info\\_get](#) (uint32\_t dv\_ind, [rsmi\\_gpu\\_metrics\\_t](#) \*pgpu\_metrics)

*This function retrieves the gpu metrics information.*
- [rsmi\\_status\\_t rsmi\\_dev\\_clk\\_range\\_set](#) (uint32\_t dv\_ind, uint64\_t minclkvalue, uint64\_t maxclkvalue, [rsmi\\_clk\\_type\\_t](#) clkType)

*This function sets the clock range information.*
- [rsmi\\_status\\_t rsmi\\_dev\\_od\\_clk\\_info\\_set](#) (uint32\_t dv\_ind, [rsmi\\_freq\\_ind\\_t](#) level, uint64\_t clkvalue, [rsmi\\_clk\\_type\\_t](#) clkType)

*This function sets the clock frequency information.*
- [rsmi\\_status\\_t rsmi\\_dev\\_od\\_volt\\_info\\_set](#) (uint32\_t dv\_ind, uint32\_t vpoint, uint64\_t clkvalue, uint64\_t volt-value)

*This function sets 1 of the 3 voltage curve points.*
- [rsmi\\_status\\_t rsmi\\_dev\\_od\\_volt\\_curve\\_regions\\_get](#) (uint32\_t dv\_ind, uint32\_t \*num\_regions, [rsmi\\_freq\\_volt\\_region\\_t](#) \*buffer)

*This function will retrieve the current valid regions in the frequency/voltage space.*
- [rsmi\\_status\\_t rsmi\\_dev\\_power\\_profile\\_presets\\_get](#) (uint32\_t dv\_ind, uint32\_t sensor\_ind, [rsmi\\_power\\_profile\\_status\\_t](#) \*status)

*Get the list of available preset power profiles and an indication of which profile is currently active.*
- [rsmi\\_status\\_t rsmi\\_dev\\_perf\\_level\\_set](#) (int32\_t dv\_ind, [rsmi\\_dev\\_perf\\_level\\_t](#) perf\_lvl)

*Set the PowerPlay performance level associated with the device with provided device index with the provided value.*
- [rsmi\\_status\\_t rsmi\\_dev\\_perf\\_level\\_set\\_v1](#) (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](#) (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\\_overdrive\\_level\\_set\\_v1](#) (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\\_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\\_firmware\\_version\\_get](#) (uint32\_t dv\_ind, [rsmi\\_fw\\_block\\_t](#) block, uint64\_t \*fw\_version)

*Get the firmware versions for a device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_ecc\\_count\\_get](#) (uint32\_t dv\_ind, [rsmi\\_gpu\\_block\\_t](#) block, [rsmi\\_error\\_count\\_t](#) \*ec)

*Retrieve the error counts for a GPU block.*
- [rsmi\\_status\\_t rsmi\\_dev\\_ecc\\_enabled\\_get](#) (uint32\_t dv\_ind, uint64\_t \*enabled\_blocks)

*Retrieve the enabled ECC bit-mask.*

- [rsmi\\_status\\_t rsmi\\_dev\\_ecc\\_status\\_get](#) (uint32\_t dv\_ind, [rsmi\\_gpu\\_block\\_t](#) block, [rsmi\\_ras\\_err\\_state\\_t](#) \*state)  
*Retrieve the ECC status for a GPU block.*
- [rsmi\\_status\\_t rsmi\\_status\\_string](#) ([rsmi\\_status\\_t](#) status, const char \*\*status\_string)  
*Get a description of a provided RSMI error status.*
- [rsmi\\_status\\_t rsmi\\_dev\\_counter\\_group\\_supported](#) (uint32\_t dv\_ind, [rsmi\\_event\\_group\\_t](#) group)  
*Tell if an event group is supported by a given device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_counter\\_create](#) (uint32\_t dv\_ind, [rsmi\\_event\\_type\\_t](#) type, [rsmi\\_event\\_handle\\_t](#) \*evnt\_handle)  
*Create a performance counter object.*
- [rsmi\\_status\\_t rsmi\\_dev\\_counter\\_destroy](#) ([rsmi\\_event\\_handle\\_t](#) evnt\_handle)  
*Deallocate a performance counter object.*
- [rsmi\\_status\\_t rsmi\\_counter\\_control](#) ([rsmi\\_event\\_handle\\_t](#) evt\_handle, [rsmi\\_counter\\_command\\_t](#) cmd, void \*cmd\_args)  
*Issue performance counter control commands.*
- [rsmi\\_status\\_t rsmi\\_counter\\_read](#) ([rsmi\\_event\\_handle\\_t](#) evt\_handle, [rsmi\\_counter\\_value\\_t](#) \*value)  
*Read the current value of a performance counter.*
- [rsmi\\_status\\_t rsmi\\_counter\\_available\\_counters\\_get](#) (uint32\_t dv\_ind, [rsmi\\_event\\_group\\_t](#) grp, uint32\_t \*available)  
*Get the number of currently available counters.*
- [rsmi\\_status\\_t rsmi\\_compute\\_process\\_info\\_get](#) ([rsmi\\_process\\_info\\_t](#) \*procs, uint32\_t \*num\_items)  
*Get process information about processes currently using GPU.*
- [rsmi\\_status\\_t rsmi\\_compute\\_process\\_info\\_by\\_pid\\_get](#) (uint32\_t pid, [rsmi\\_process\\_info\\_t](#) \*proc)  
*Get process information about a specific process.*
- [rsmi\\_status\\_t rsmi\\_compute\\_process\\_gpus\\_get](#) (uint32\_t pid, uint32\_t \*dv\_indices, uint32\_t \*num\_devices)  
*Get the device indices currently being used by a process.*
- [rsmi\\_status\\_t rsmi\\_dev\\_xgmi\\_error\\_status](#) (uint32\_t dv\_ind, [rsmi\\_xgmi\\_status\\_t](#) \*status)  
*Retrieve the XGMI error status for a device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_xgmi\\_error\\_reset](#) (uint32\_t dv\_ind)  
*Reset the XGMI error status for a device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_xgmi\\_hive\\_id\\_get](#) (uint32\_t dv\_ind, uint64\_t \*hive\_id)  
*Retrieve the XGMI hive id for a device.*
- [rsmi\\_status\\_t rsmi\\_topo\\_get\\_numa\\_node\\_number](#) (uint32\_t dv\_ind, uint32\_t \*numa\_node)  
*Retrieve the NUMA CPU node number for a device.*
- [rsmi\\_status\\_t rsmi\\_topo\\_get\\_link\\_weight](#) (uint32\_t dv\_ind\_src, uint32\_t dv\_ind\_dst, uint64\_t \*weight)  
*Retrieve the weight for a connection between 2 GPUs.*
- [rsmi\\_status\\_t rsmi\\_minmax\\_bandwidth\\_get](#) (uint32\_t dv\_ind\_src, uint32\_t dv\_ind\_dst, uint64\_t \*min\_bandwidth, uint64\_t \*max\_bandwidth)  
*Retrieve minimal and maximal io link bandwidth between 2 GPUs.*
- [rsmi\\_status\\_t rsmi\\_topo\\_get\\_link\\_type](#) (uint32\_t dv\_ind\_src, uint32\_t dv\_ind\_dst, uint64\_t \*hops, [RSMI\\_IO\\_LINK\\_TYPE](#) \*type)  
*Retrieve the hops and the connection type between 2 GPUs.*
- [rsmi\\_status\\_t rsmi\\_is\\_P2P\\_accessible](#) (uint32\_t dv\_ind\_src, uint32\_t dv\_ind\_dst, bool \*accessible)  
*Return P2P availability status between 2 GPUs.*
- [rsmi\\_status\\_t rsmi\\_dev\\_compute\\_partition\\_get](#) (uint32\_t dv\_ind, char \*compute\_partition, uint32\_t len)  
*Retrieves the current compute partitioning for a desired device.*
- [rsmi\\_status\\_t rsmi\\_dev\\_compute\\_partition\\_set](#) (uint32\_t dv\_ind, [rsmi\\_compute\\_partition\\_type\\_t](#) compute\_partition)  
*Modifies a selected device's compute partition setting.*
- [rsmi\\_status\\_t rsmi\\_dev\\_compute\\_partition\\_reset](#) (uint32\_t dv\_ind)  
*Reverts a selected device's compute partition setting back to its boot state.*
- [rsmi\\_status\\_t rsmi\\_dev\\_nps\\_mode\\_get](#) (uint32\_t dv\_ind, char \*nps\_mode, uint32\_t len)

- Retrieves the NPS mode (memory partition) for a desired device.*
- `rsmi_status_t rsmi_dev_nps_mode_set` (uint32\_t dv\_ind, `rsmi_nps_mode_type_t` nps\_mode)
- Modifies a selected device's NPS mode (memory partition) setting.*
- `rsmi_status_t rsmi_dev_nps_mode_reset` (uint32\_t dv\_ind)
- Reverts a selected device's NPS mode setting back to its boot state.*
- `rsmi_status_t rsmi_dev_supported_func_iterator_open` (uint32\_t dv\_ind, `rsmi_func_id_iter_handle_t` \*handle)
- Get a function name iterator of supported RSMI functions for a device.*
- `rsmi_status_t rsmi_dev_supported_variant_iterator_open` (`rsmi_func_id_iter_handle_t` obj\_h, `rsmi_func_id_iter_handle_t` \*var\_iter)
- Get a variant iterator for a given handle.*
- `rsmi_status_t rsmi_func_iter_next` (`rsmi_func_id_iter_handle_t` handle)
- Advance a function identifier iterator.*
- `rsmi_status_t rsmi_dev_supported_func_iterator_close` (`rsmi_func_id_iter_handle_t` \*handle)
- Close a variant iterator handle.*
- `rsmi_status_t rsmi_func_iter_value_get` (`rsmi_func_id_iter_handle_t` handle, `rsmi_func_id_value_t` \*value)
- Get the value associated with a function/variant iterator.*
- `rsmi_status_t rsmi_event_notification_init` (uint32\_t dv\_ind)
- Prepare to collect event notifications for a GPU.*
- `rsmi_status_t rsmi_event_notification_mask_set` (uint32\_t dv\_ind, uint64\_t mask)
- Specify which events to collect for a device.*
- `rsmi_status_t rsmi_event_notification_get` (int timeout\_ms, uint32\_t \*num\_elem, `rsmi_evt_notification_data_t` \*data)
- Collect event notifications, waiting a specified amount of time.*
- `rsmi_status_t rsmi_event_notification_stop` (uint32\_t dv\_ind)
- Close any file handles and free any resources used by event notification for a GPU.*

### 8.1.1 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.

### 8.1.2 Macro Definition Documentation

#### 8.1.2.1 RSMI\_MAX\_FAN\_SPEED

```
#define RSMI_MAX_FAN_SPEED 255
```

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

#### 8.1.2.2 RSMI\_EVENT\_MASK\_FROM\_INDEX

```
#define RSMI_EVENT_MASK_FROM_INDEX(  
    i ) (1ULL << ((i) - 1))
```

Macro to generate event bitmask from event id

### 8.1.2.3 RSMI\_DEFAULT\_VARIANT

```
#define RSMI_DEFAULT_VARIANT 0xFFFFFFFFFFFFFFFF
```

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

## 8.1.3 Typedef Documentation

### 8.1.3.1 rsmi\_event\_handle\_t

```
typedef uintptr_t rsmi_event_handle_t
```

Handle to performance event counter.

Event counter types

## 8.1.4 Enumeration Type Documentation

### 8.1.4.1 rsmi\_status\_t

```
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 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_NO_DATA             | No data was found for a given input                                                                                                    |
| RSMI_STATUS_UNEXPECTED_DATA     | The data read or provided to function is not what was expected                                                                         |
| RSMI_STATUS_BUSY                | A resource or mutex could not be acquired because it is already being used                                                             |
| RSMI_STATUS_REFCOUNT_OVERFLOW   | An internal reference counter exceeded INT32_MAX                                                                                       |
| RSMI_STATUS_SETTING_UNAVAILABLE | Requested setting is unavailable for the current device                                                                                |
| RSMI_STATUS_AMDGPU_RESTART_ERR  | Could not successfully restart the amdgpu driver                                                                                       |
| RSMI_STATUS_UNKNOWN_ERROR       | An unknown error occurred.                                                                                                             |

### 8.1.4.2 `rsmi_init_flags_t`

enum `rsmi_init_flags_t`

Initialization flags.

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

#### Enumerator

|                                         |                                                                                                                                                                            |
|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>RSMI_INIT_FLAG_ALL_GPUS</code>    | 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. |
| <code>RSMI_INIT_FLAG_RESRV_TEST1</code> | Reserved for test.                                                                                                                                                         |

### 8.1.4.3 `rsmi_dev_perf_level_t`

enum `rsmi_dev_perf_level_t`

PowerPlay performance levels.

#### Enumerator

|                                                  |                                                                                      |
|--------------------------------------------------|--------------------------------------------------------------------------------------|
| <code>RSMI_DEV_PERF_LEVEL_AUTO</code>            | Performance level is "auto".                                                         |
| <code>RSMI_DEV_PERF_LEVEL_LOW</code>             | Keep PowerPlay levels "low", regardless of workload                                  |
| <code>RSMI_DEV_PERF_LEVEL_HIGH</code>            | Keep PowerPlay levels "high", regardless of workload                                 |
| <code>RSMI_DEV_PERF_LEVEL_MANUAL</code>          | Only use values defined by manually setting the <code>RSMI_CLK_TYPE_SYS</code> speed |
| <code>RSMI_DEV_PERF_LEVEL_STABLE_STD</code>      | Stable power state with profiling clocks                                             |
| <code>RSMI_DEV_PERF_LEVEL_STABLE_PEAK</code>     | Stable power state with peak clocks.                                                 |
| <code>RSMI_DEV_PERF_LEVEL_STABLE_MIN_MCLK</code> | Stable power state with minimum memory clock                                         |
| <code>RSMI_DEV_PERF_LEVEL_STABLE_MIN_SCLK</code> | Stable power state with minimum system clock                                         |
| <code>RSMI_DEV_PERF_LEVEL_DETERMINISM</code>     | Performance determinism state.                                                       |
| <code>RSMI_DEV_PERF_LEVEL_UNKNOWN</code>         | Unknown performance level.                                                           |

### 8.1.4.4 `rsmi_sw_component_t`

enum `rsmi_sw_component_t`

Available clock types.

Software components

#### Enumerator

|                                  |         |
|----------------------------------|---------|
| <code>RSMI_SW_COMP_DRIVER</code> | Driver. |
|----------------------------------|---------|

#### 8.1.4.5 rsmi\_event\_group\_t

```
enum rsmi_event_group_t
```

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

Event Groups

Enumerator

|                             |                                    |
|-----------------------------|------------------------------------|
| RSMI_EVNT_GRP_XGMI          | Data Fabric (XGMI) related events. |
| RSMI_EVNT_GRP_XGMI_DATA_OUT | XGMI Outbound data.                |

#### 8.1.4.6 rsmi\_event\_type\_t

```
enum rsmi_event_type_t
```

Event type enum. Events belonging to a particular event group [rsmi\\_event\\_group\\_t](#) should begin 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    | <p>Data beats sent to neighbor 0; Each beat represents 32 bytes.</p> <p>XGMI throughput can be calculated by multiplying a BEATs event such as <a href="#">RSMI_EVNT_XGMI_0_BEATS_TX</a> by 32 and dividing by the time for which event collection occurred, <a href="#">rsmi_counter_value_t.time_running</a> (which is in nanoseconds). To get bytes per second, multiply this value by <math>10^9</math>.</p> <p>Throughput = BEATS/time_running * <math>10^9</math> (bytes/second)</p> |
| RSMI_EVNT_XGMI_1_NOP_TX      | NOPs sent to neighbor 1.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| RSMI_EVNT_XGMI_1_REQUEST_TX  | neighbor 1 Outgoing requests to                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| RSMI_EVNT_XGMI_1_RESPONSE_TX | Outgoing responses to neighbor 1                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| RSMI_EVNT_XGMI_1_BEATS_TX    | Data beats sent to neighbor 1; Each beat represents 32 bytes                                                                                                                                                                                                                                                                                                                                                                                                                               |
| RSMI_EVNT_XGMI_DATA_OUT_1    | Outbound beats to neighbor 1.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| RSMI_EVNT_XGMI_DATA_OUT_2    | Outbound beats to neighbor 2.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| RSMI_EVNT_XGMI_DATA_OUT_3    | Outbound beats to neighbor 3.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| RSMI_EVNT_XGMI_DATA_OUT_4    | Outbound beats to neighbor 4.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| RSMI_EVNT_XGMI_DATA_OUT_5    | Outbound beats to neighbor 5.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

#### 8.1.4.7 rsmi\_counter\_command\_t

```
enum rsmi_counter_command_t
```

Event counter commands

Enumerator

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| RSMI_CNTR_CMD_START | Start the counter.                                                  |
| RSMI_CNTR_CMD_STOP  | Stop the counter; note that this should not be used before reading. |

#### 8.1.4.8 rsmi\_evt\_notification\_type\_t

enum `rsmi_evt_notification_type_t`

Event notification event types

Enumerator

|                        |                |
|------------------------|----------------|
| RSMI_EVT_NOTIF_VMFAULT | VM page fault. |
|------------------------|----------------|

#### 8.1.4.9 rsmi\_clk\_type\_t

enum `rsmi_clk_type_t`

Clock types

Enumerator

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

#### 8.1.4.10 rsmi\_compute\_partition\_type\_t

enum `rsmi_compute_partition_type_t`

Compute Partition. This enum is used to identify various compute partitioning settings.

Enumerator

|                            |                                                                        |
|----------------------------|------------------------------------------------------------------------|
| RSMI_COMPUTE_PARTITION_CPX | Core mode (CPX)- Per-chip XCC with shared memory                       |
| RSMI_COMPUTE_PARTITION_SPX | Single GPU mode (SPX)- All XCCs work together with shared memory       |
| RSMI_COMPUTE_PARTITION_DPX | Dual GPU mode (DPX)- Half XCCs work together with shared memory        |
| RSMI_COMPUTE_PARTITION_TPX | Triple GPU mode (TPX)- One-third XCCs work together with shared memory |
| RSMI_COMPUTE_PARTITION_QPX | Quad GPU mode (QPX)- Quarter XCCs work together with shared memory     |

## 8.1.4.11 rsmi\_nps\_mode\_type\_t

```
enum rsmi_nps_mode_type_t
```

NPS Modes. This enum is used to identify various NPS mode types.

## Enumerator

|                            |                                                                                                                                                                           |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RSMI_MEMORY_PARTITION_NPS1 | NPS1 - All CCD & XCD data is interleaved accross all 8 HBM stacks (all stacks/1).                                                                                         |
| RSMI_MEMORY_PARTITION_NPS2 | NPS2 - 2 sets of CCDs or 4 XCD interleaved accross the 4 HBM stacks per AID pair (8 stacks/2).                                                                            |
| RSMI_MEMORY_PARTITION_NPS4 | NPS4 - Each XCD data is interleaved accross accross 2 (or single) HBM stacks (8 stacks/8 or 8 stacks/4).                                                                  |
| RSMI_MEMORY_PARTITION_NPS8 | NPS8 - Each XCD uses a single HBM stack (8 stacks/8). Or each XCD uses a single HBM stack & CCDs share 2 non-interleaved HBM stacks on its AID (AID[1,2,3] = 6 stacks/6). |

## 8.1.4.12 rsmi\_temperature\_metric\_t

```
enum rsmi_temperature_metric_t
```

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

## Enumerator

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

#### 8.1.4.13 rsmi\_temperature\_type\_t

```
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.            |
| RSMI_TEMP_TYPE_HBM_0    | HBM temperature instance 0.  |
| RSMI_TEMP_TYPE_HBM_1    | HBM temperature instance 1.  |
| RSMI_TEMP_TYPE_HBM_2    | HBM temperature instance 2.  |
| RSMI_TEMP_TYPE_HBM_3    | HBM temperature instance 3.  |
| RSMI_TEMP_TYPE_INVALID  | Invalid type.                |

#### 8.1.4.14 rsmi\_voltage\_metric\_t

```
enum rsmi_voltage_metric_t
```

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

##### Enumerator

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

#### 8.1.4.15 rsmi\_voltage\_type\_t

```
enum rsmi_voltage_type_t
```

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

##### Enumerator

|                        |                    |
|------------------------|--------------------|
| RSMI_VOLT_TYPE_VDDGFX  | Vddgfx GPU voltage |
| RSMI_VOLT_TYPE_INVALID | Invalid type.      |

**8.1.4.16 rsmi\_power\_profile\_preset\_masks\_t**

```
enum rsmi_power_profile_preset_masks_t
```

Pre-set Profile Selections. These bitmasks can be AND'd with the [rsmi\\_power\\_profile\\_status\\_t.available\\_profiles](#) returned from [rsmi\\_dev\\_power\\_profile\\_presets\\_get](#) to determine which power profiles are supported by the system.

**Enumerator**

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

**8.1.4.17 rsmi\_gpu\_block\_t**

```
enum rsmi_gpu_block_t
```

This enum is used to identify different GPU blocks.

**Enumerator**

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

**8.1.4.18 rsmi\_ras\_err\_state\_t**

```
enum rsmi_ras_err_state_t
```

The current ECC state.

## Enumerator

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

## 8.1.4.19 rsmi\_memory\_type\_t

```
enum rsmi_memory_type_t
```

Types of memory.

## Enumerator

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

## 8.1.4.20 rsmi\_freq\_ind\_t

```
enum rsmi_freq_ind_t
```

The values of this enum are used as frequency identifiers.

## Enumerator

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

## 8.1.4.21 rsmi\_memory\_page\_status\_t

```
enum rsmi_memory_page_status_t
```

Reserved Memory Page States.

## Enumerator

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

## 8.1.4.22 \_RSMI\_IO\_LINK\_TYPE

```
enum _RSMI_IO_LINK_TYPE
```

Types for IO Link.

## Enumerator

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

## 8.1.4.23 RSMI\_UTILIZATION\_COUNTER\_TYPE

```
enum RSMI_UTILIZATION_COUNTER_TYPE
```

The utilization counter type.

## Enumerator

|                                |                  |
|--------------------------------|------------------|
| RSMI_UTILIZATION_COUNTER_FIRST | GFX Activity.    |
| RSMI_COARSE_GRAIN_MEM_ACTIVITY | Memory Activity. |

## 8.2 rocm\_smi.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * =====
00003  * The University of Illinois/NCSA
00004  * Open Source License (NCSA)
00005  *
00006  * Copyright (c) 2017-2023, Advanced Micro Devices, Inc.
00007  * All rights reserved.
00008  *
00009  * Developed by:
00010  *
00011  *          AMD Research and AMD ROC Software Development
00012  *
00013  *          Advanced Micro Devices, Inc.
00014  *
00015  *          www.amd.com
00016  *
00017  * Permission is hereby granted, free of charge, to any person obtaining a copy
00018  * of this software and associated documentation files (the "Software"), to
00019  * deal with the Software without restriction, including without limitation
00020  * the rights to use, copy, modify, merge, publish, distribute, sublicense,
00021  * and/or sell copies of the Software, and to permit persons to whom the
00022  * Software is furnished to do so, subject to the following conditions:
00023  *
00024  * - Redistributions of source code must retain the above copyright notice,
00025  *   this list of conditions and the following disclaimers.
00026  * - Redistributions in binary form must reproduce the above copyright
00027  *   notice, this list of conditions and the following disclaimers in
00028  *   the documentation and/or other materials provided with the distribution.
00029  * - Neither the names of <Name of Development Group, Name of Institution>,
00030  *   nor the names of its contributors may be used to endorse or promote
00031  *   products derived from this Software without specific prior written
00032  *   permission.
```

```

00033 *
00034 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00035 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00036 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
00037 * THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
00038 * OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
00039 * ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
00040 * DEALINGS WITH THE SOFTWARE.
00041 *
00042 */
00043 #ifndef INCLUDE_ROCM_SMI_ROCM_SMI_H_
00044 #define INCLUDE_ROCM_SMI_ROCM_SMI_H_
00045
00046 #ifdef __cplusplus
00047 extern "C" {
00048 #include <stdint>
00049 #else
00050 #include <stdint.h>
00051 #endif // __cplusplus
00052
00053 #include <stddef.h>
00054 #include <stdbool.h>
00055
00056 #include "rocm_smi/kfd_ioctl.h"
00057
00073 #define RSMI_MAX_NUM_FREQUENCIES 32
00074
00077 #define RSMI_MAX_FAN_SPEED 255
00078
00080 #define RSMI_NUM_VOLTAGE_CURVE_POINTS 3
00081
00085 typedef enum {
00086     RSMI_STATUS_SUCCESS = 0x0,
00087     RSMI_STATUS_INVALID_ARGS,
00088     RSMI_STATUS_NOT_SUPPORTED,
00091     RSMI_STATUS_FILE_ERROR,
00096     RSMI_STATUS_PERMISSION,
00099     RSMI_STATUS_OUT_OF_RESOURCES,
0101     RSMI_STATUS_INTERNAL_EXCEPTION,
0102     RSMI_STATUS_INPUT_OUT_OF_BOUNDS,
0104     RSMI_STATUS_INIT_ERROR,
0107     RSMI_INITIALIZATION_ERROR = RSMI_STATUS_INIT_ERROR,
0108     RSMI_STATUS_NOT_YET_IMPLEMENTED,
0112     RSMI_STATUS_NOT_FOUND,
0114     RSMI_STATUS_INSUFFICIENT_SIZE,
0116     RSMI_STATUS_INTERRUPT,
0118     RSMI_STATUS_UNEXPECTED_SIZE,
0120     RSMI_STATUS_NO_DATA,
0122     RSMI_STATUS_UNEXPECTED_DATA,
0124     RSMI_STATUS_BUSY,
0127     RSMI_STATUS_REFCOUNT_OVERFLOW,
0129     RSMI_STATUS_SETTING_UNAVAILABLE,
0131     RSMI_STATUS_AMDGPU_RESTART_ERR,
0133
0134     RSMI_STATUS_UNKNOWN_ERROR = 0xFFFFFFFF,
0135 } rsmi_status_t;
0136
0143 typedef enum {
0144     RSMI_INIT_FLAG_ALL_GPUS = 0x1,
0150     RSMI_INIT_FLAG_RESRV_TEST1 = 0x8000000000000000,
0151 } rsmi_init_flags_t;
0152
0156 typedef enum {
0157     RSMI_DEV_PERF_LEVEL_AUTO = 0,
0158     RSMI_DEV_PERF_LEVEL_FIRST = RSMI_DEV_PERF_LEVEL_AUTO,
0159
0160     RSMI_DEV_PERF_LEVEL_LOW,
0162     RSMI_DEV_PERF_LEVEL_HIGH,
0164     RSMI_DEV_PERF_LEVEL_MANUAL,
0166     RSMI_DEV_PERF_LEVEL_STABLE_STD,
0168     RSMI_DEV_PERF_LEVEL_STABLE_PEAK,
0169     RSMI_DEV_PERF_LEVEL_STABLE_MIN_MCLK,
0171     RSMI_DEV_PERF_LEVEL_STABLE_MIN_SCLK,
0173     RSMI_DEV_PERF_LEVEL_DETERMINISM,
0174
0175     RSMI_DEV_PERF_LEVEL_LAST = RSMI_DEV_PERF_LEVEL_DETERMINISM,
0176
0177     RSMI_DEV_PERF_LEVEL_UNKNOWN = 0x100
0178 } rsmi_dev_perf_level_t;
0180 typedef rsmi_dev_perf_level_t rsmi_dev_perf_level;
0182
0189 typedef enum {
0190     RSMI_SW_COMP_FIRST = 0x0,
0191
0192     RSMI_SW_COMP_DRIVER = RSMI_SW_COMP_FIRST,
0193
0194     RSMI_SW_COMP_LAST = RSMI_SW_COMP_DRIVER

```

```

00195 } rsmi_sw_component_t;
00196
00204 typedef uintptr_t rsmi_event_handle_t;
00205
00212 typedef enum {
00213     RSMI_EVT_GRP_XGMI = 0,
00214     RSMI_EVT_GRP_XGMI_DATA_OUT = 10,
00215     RSMI_EVT_GRP_INVALID = 0xFFFFFFFF
00216 } rsmi_event_group_t;
00217
00224 typedef enum {
00225     RSMI_EVT_FIRST = RSMI_EVT_GRP_XGMI,
00226
00227     RSMI_EVT_XGMI_FIRST = RSMI_EVT_GRP_XGMI,
00228     RSMI_EVT_XGMI_0_NOP_TX = RSMI_EVT_XGMI_FIRST,
00229     RSMI_EVT_XGMI_0_REQUEST_TX,
00231     RSMI_EVT_XGMI_0_RESPONSE_TX,
00233
00246     // ie, Throughput = BEATS/time_running 10^9 bytes/sec
00247     RSMI_EVT_XGMI_0_BEATS_TX,
00248     RSMI_EVT_XGMI_1_NOP_TX,
00249     RSMI_EVT_XGMI_1_REQUEST_TX,
00251     RSMI_EVT_XGMI_1_RESPONSE_TX,
00253     RSMI_EVT_XGMI_1_BEATS_TX,
00256
00257     RSMI_EVT_XGMI_LAST = RSMI_EVT_XGMI_1_BEATS_TX,    // 5
00258
00259     RSMI_EVT_XGMI_DATA_OUT_FIRST = RSMI_EVT_GRP_XGMI_DATA_OUT,    // 10
00260
00261     /*
00262      * @brief Events in the RSMI_EVT_GRP_XGMI_DATA_OUT group measure
00263      * the number of beats sent on an XGMI link. Each beat represents
00264      * 32 bytes. RSMI_EVT_XGMI_DATA_OUT_n represents the number of
00265      * outbound beats (each representing 32 bytes) on link n.<br><br>
00266      *
00267      * XGMI throughput can be calculated by multiplying a event
00268      * such as ::RSMI_EVT_XGMI_DATA_OUT_n by 32 and dividing by
00269      * the time for which event collection occurred,
00270      * ::rsmi_counter_value_t.time_running (which is in nanoseconds). To get
00271      * bytes per second, multiply this value by 10<sup>9</sup><br>
00272      * <br>
00273      * Throughput = BEATS/time_running * 10<sup>9</sup> (bytes/second)<br>
00274      */
00275     // ie, Throughput = BEATS/time_running 10^9 bytes/sec
00276     RSMI_EVT_XGMI_DATA_OUT_0 = RSMI_EVT_XGMI_DATA_OUT_FIRST,
00277     RSMI_EVT_XGMI_DATA_OUT_1,
00278     RSMI_EVT_XGMI_DATA_OUT_2,
00279     RSMI_EVT_XGMI_DATA_OUT_3,
00280     RSMI_EVT_XGMI_DATA_OUT_4,
00281     RSMI_EVT_XGMI_DATA_OUT_5,
00282     RSMI_EVT_XGMI_DATA_OUT_LAST = RSMI_EVT_XGMI_DATA_OUT_5,
00283
00284     RSMI_EVT_LAST = RSMI_EVT_XGMI_DATA_OUT_LAST,
00285 } rsmi_event_type_t;
00286
00290 typedef enum {
00291     RSMI_CNTR_CMD_START = 0,
00292     RSMI_CNTR_CMD_STOP,
00294 } rsmi_counter_command_t;
00295
00299 typedef struct {
00300     uint64_t value;
00301     uint64_t time_enabled;
00303     uint64_t time_running;
00305 } rsmi_counter_value_t;
00306
00310 typedef enum {
00311     RSMI_EVT_NOTIF_VMFALUT = KFD_SMI_EVENT_VMFALUT,
00312     RSMI_EVT_NOTIF_FIRST = RSMI_EVT_NOTIF_VMFALUT,
00313     RSMI_EVT_NOTIF_THERMAL_THROTTLE = KFD_SMI_EVENT_THERMAL_THROTTLE,
00314     RSMI_EVT_NOTIF_GPU_PRE_RESET = KFD_SMI_EVENT_GPU_PRE_RESET,
00315     RSMI_EVT_NOTIF_GPU_POST_RESET = KFD_SMI_EVENT_GPU_POST_RESET,
00316
00317     RSMI_EVT_NOTIF_LAST = RSMI_EVT_NOTIF_GPU_POST_RESET
00318 } rsmi_evt_notification_type_t;
00319
00323 #define RSMI_EVENT_MASK_FROM_INDEX(i) (1ULL << ((i) - 1))
00324
00326 #define MAX_EVENT_NOTIFICATION_MSG_SIZE 64
00327
00331 typedef struct {
00332     uint32_t dv_ind;
00333     rsmi_evt_notification_type_t event;
00334     char message[MAX_EVENT_NOTIFICATION_MSG_SIZE];
00335 } rsmi_evt_notification_data_t;
00336
00340 typedef enum {

```

```

00341     RSMI_CLK_TYPE_SYS = 0x0,
00342     RSMI_CLK_TYPE_FIRST = RSMI_CLK_TYPE_SYS,
00343     RSMI_CLK_TYPE_DF,
00344     RSMI_CLK_TYPE_DCEF,
00345     RSMI_CLK_TYPE_SOC,
00346     RSMI_CLK_TYPE_MEM,
00347     RSMI_CLK_TYPE_PCIE,
00348
00349
00350     // Add new clocks to the end (not in the middle) and update
00351     // RSMI_CLK_TYPE_LAST
00352     RSMI_CLK_TYPE_LAST = RSMI_CLK_TYPE_MEM,
00353     RSMI_CLK_INVALID = 0xFFFFFFFF
00354 } rsmi_clk_type_t;
00355 typedef rsmi_clk_type_t rsmi_clk_type;
00356
00357 typedef enum {
00358     RSMI_COMPUTE_PARTITION_INVALID = 0,
00359     RSMI_COMPUTE_PARTITION_CPX,
00360     RSMI_COMPUTE_PARTITION_SPX,
00361     RSMI_COMPUTE_PARTITION_DPX,
00362     RSMI_COMPUTE_PARTITION_TPX,
00363     RSMI_COMPUTE_PARTITION_QPX
00364 } rsmi_compute_partition_type_t;
00365 typedef rsmi_compute_partition_type_t rsmi_compute_partition_type;
00366
00367 typedef enum {
00368     RSMI_MEMORY_PARTITION_UNKNOWN = 0,
00369     RSMI_MEMORY_PARTITION_NPS1,
00370     RSMI_MEMORY_PARTITION_NPS2,
00371     RSMI_MEMORY_PARTITION_NPS4,
00372     RSMI_MEMORY_PARTITION_NPS8,
00373 } rsmi_nps_mode_type_t;
00374 typedef rsmi_nps_mode_type_t rsmi_nps_mode_type;
00375
00376 typedef enum {
00377     RSMI_TEMP_CURRENT = 0x0,
00378     RSMI_TEMP_FIRST = RSMI_TEMP_CURRENT,
00379
00380     RSMI_TEMP_MAX,
00381     RSMI_TEMP_MIN,
00382     RSMI_TEMP_MAX_HYST,
00383     RSMI_TEMP_MIN_HYST,
00384     RSMI_TEMP_CRITICAL,
00385     RSMI_TEMP_CRITICAL_HYST,
00386     RSMI_TEMP_EMERGENCY,
00387     RSMI_TEMP_EMERGENCY_HYST,
00388     RSMI_TEMP_CRIT_MIN,
00389     RSMI_TEMP_CRIT_MIN_HYST,
00390     RSMI_TEMP_OFFSET,
00391     RSMI_TEMP_LOWEST,
00392     RSMI_TEMP_HIGHEST,
00393
00394     RSMI_TEMP_LAST = RSMI_TEMP_HIGHEST
00395 } rsmi_temperature_metric_t;
00396 typedef rsmi_temperature_metric_t rsmi_temperature_metric;
00397
00398 typedef enum {
00399     RSMI_TEMP_TYPE_FIRST = 0,
00400
00401     RSMI_TEMP_TYPE_EDGE = RSMI_TEMP_TYPE_FIRST,
00402     RSMI_TEMP_TYPE_JUNCTION,
00403     RSMI_TEMP_TYPE_MEMORY,
00404     RSMI_TEMP_TYPE_HBM_0,
00405     RSMI_TEMP_TYPE_HBM_1,
00406     RSMI_TEMP_TYPE_HBM_2,
00407     RSMI_TEMP_TYPE_HBM_3,
00408     RSMI_TEMP_TYPE_LAST = RSMI_TEMP_TYPE_HBM_3,
00409     RSMI_TEMP_TYPE_INVALID = 0xFFFFFFFF
00410 } rsmi_temperature_type_t;
00411
00412 typedef enum {
00413     RSMI_VOLT_CURRENT = 0x0,
00414
00415     RSMI_VOLT_FIRST = RSMI_VOLT_CURRENT,
00416     RSMI_VOLT_MAX,
00417     RSMI_VOLT_MIN_CRIT,
00418     RSMI_VOLT_MIN,
00419     RSMI_VOLT_MAX_CRIT,
00420     RSMI_VOLT_AVERAGE,
00421     RSMI_VOLT_LOWEST,
00422     RSMI_VOLT_HIGHEST,
00423
00424     RSMI_VOLT_LAST = RSMI_VOLT_HIGHEST
00425 } rsmi_voltage_metric_t;
00426
00427 typedef enum {
00428     RSMI_VOLT_TYPE_FIRST = 0,

```

```

00495
00496     RSMI_VOLT_TYPE_VDDGFX = RSMI_VOLT_TYPE_FIRST,
00498     RSMI_VOLT_TYPE_LAST = RSMI_VOLT_TYPE_VDDGFX,
00499     RSMI_VOLT_TYPE_INVALID = 0xFFFFFFFF
00500 } rsmi_voltage_type_t;
00501
00508 typedef enum {
00509     RSMI_PWR_PROF_PRST_CUSTOM_MASK = 0x1,
00510     RSMI_PWR_PROF_PRST_VIDEO_MASK = 0x2,
00511     RSMI_PWR_PROF_PRST_POWER_SAVING_MASK = 0x4,
00512     RSMI_PWR_PROF_PRST_COMPUTE_MASK = 0x8,
00513     RSMI_PWR_PROF_PRST_VR_MASK = 0x10,
00514
00516     RSMI_PWR_PROF_PRST_3D_FULL_SCR_MASK = 0x20,
00517     RSMI_PWR_PROF_PRST_BOOTUP_DEFAULT = 0x40,
00518     RSMI_PWR_PROF_PRST_LAST = RSMI_PWR_PROF_PRST_BOOTUP_DEFAULT,
00519
00521     RSMI_PWR_PROF_PRST_INVALID = 0xFFFFFFFFFFFFFFFF
00522 } rsmi_power_profile_preset_masks_t;
00524 typedef rsmi_power_profile_preset_masks_t rsmi_power_profile_preset_masks;
00526
00530 typedef enum {
00531     RSMI_GPU_BLOCK_INVALID = 0x0000000000000000,
00533     RSMI_GPU_BLOCK_FIRST = 0x0000000000000001,
00534
00536     RSMI_GPU_BLOCK_UMC = RSMI_GPU_BLOCK_FIRST,
00536     RSMI_GPU_BLOCK_SDMA = 0x0000000000000002,
00537     RSMI_GPU_BLOCK_GFX = 0x0000000000000004,
00538     RSMI_GPU_BLOCK_MMHUB = 0x0000000000000008,
00539     RSMI_GPU_BLOCK_ATHUB = 0x0000000000000010,
00540     RSMI_GPU_BLOCK_PCIE_BIF = 0x0000000000000020,
00541     RSMI_GPU_BLOCK_HDP = 0x0000000000000040,
00542     RSMI_GPU_BLOCK_XGMI_WAFL = 0x0000000000000080,
00543     RSMI_GPU_BLOCK_DF = 0x0000000000000100,
00544     RSMI_GPU_BLOCK_SMN = 0x0000000000000200,
00545     RSMI_GPU_BLOCK_SEM = 0x0000000000000400,
00546     RSMI_GPU_BLOCK_MPO = 0x0000000000000800,
00547     RSMI_GPU_BLOCK_MPL = 0x0000000000001000,
00548     RSMI_GPU_BLOCK_FUSE = 0x0000000000002000,
00549
00550     RSMI_GPU_BLOCK_LAST = RSMI_GPU_BLOCK_FUSE,
00552     RSMI_GPU_BLOCK_RESERVED = 0x8000000000000000
00553 } rsmi_gpu_block_t;
00555 typedef rsmi_gpu_block_t rsmi_gpu_block;
00557
00561 typedef enum {
00562     RSMI_RAS_ERR_STATE_NONE = 0,
00563     RSMI_RAS_ERR_STATE_DISABLED,
00564     RSMI_RAS_ERR_STATE_PARITY,
00565     RSMI_RAS_ERR_STATE_SING_C,
00566     RSMI_RAS_ERR_STATE_MULT_UC,
00567     RSMI_RAS_ERR_STATE_POISON,
00569     RSMI_RAS_ERR_STATE_ENABLED,
00570
00571     RSMI_RAS_ERR_STATE_LAST = RSMI_RAS_ERR_STATE_ENABLED,
00572     RSMI_RAS_ERR_STATE_INVALID = 0xFFFFFFFF
00573 } rsmi_ras_err_state_t;
00574
00578 typedef enum {
00579     RSMI_MEM_TYPE_FIRST = 0,
00580
00581     RSMI_MEM_TYPE_VRAM = RSMI_MEM_TYPE_FIRST,
00582     RSMI_MEM_TYPE_VIS_VRAM,
00583     RSMI_MEM_TYPE_GTT,
00584
00585     RSMI_MEM_TYPE_LAST = RSMI_MEM_TYPE_GTT
00586 } rsmi_memory_type_t;
00587
00591 typedef enum {
00592     RSMI_FREQ_IND_MIN = 0,
00593     RSMI_FREQ_IND_MAX = 1,
00594     RSMI_FREQ_IND_INVALID = 0xFFFFFFFF
00595 } rsmi_freq_ind_t;
00597 typedef rsmi_freq_ind_t rsmi_freq_ind;
00599
00605 typedef enum {
00606     RSMI_FW_BLOCK_FIRST = 0,
00607
00608     RSMI_FW_BLOCK_ASD = RSMI_FW_BLOCK_FIRST,
00609     RSMI_FW_BLOCK_CE,
00610     RSMI_FW_BLOCK_DMCU,
00611     RSMI_FW_BLOCK_MC,
00612     RSMI_FW_BLOCK_ME,
00613     RSMI_FW_BLOCK_MEC,
00614     RSMI_FW_BLOCK_MEC2,
00615     RSMI_FW_BLOCK_PFP,

```

```

00616     RSMI_FW_BLOCK_RLC,
00617     RSMI_FW_BLOCK_RLC_SRLC,
00618     RSMI_FW_BLOCK_RLC_SRLG,
00619     RSMI_FW_BLOCK_RLC_SRLS,
00620     RSMI_FW_BLOCK_SDMA,
00621     RSMI_FW_BLOCK_SDMA2,
00622     RSMI_FW_BLOCK_SMC,
00623     RSMI_FW_BLOCK_SOS,
00624     RSMI_FW_BLOCK_TA_RAS,
00625     RSMI_FW_BLOCK_TA_XGMI,
00626     RSMI_FW_BLOCK_UVD,
00627     RSMI_FW_BLOCK_VCE,
00628     RSMI_FW_BLOCK_VCN,
00629
00630     RSMI_FW_BLOCK_LAST = RSMI_FW_BLOCK_VCN
00631 } rsmi_fw_block_t;
00632
00636 typedef enum {
00637     RSMI_XGMI_STATUS_NO_ERRORS = 0,
00638     RSMI_XGMI_STATUS_ERROR,
00639     RSMI_XGMI_STATUS_MULTIPLE_ERRORS,
00640 } rsmi_xgmi_status_t;
00641
00645 typedef uint64_t rsmi_bit_field_t;
00647 typedef rsmi_bit_field_t rsmi_bit_field;
00649
00653 typedef enum {
00654     RSMI_MEM_PAGE_STATUS_RESERVED = 0,
00656     RSMI_MEM_PAGE_STATUS_PENDING,
00659     RSMI_MEM_PAGE_STATUS_UNRESERVABLE
00660 } rsmi_memory_page_status_t;
00661
00665 typedef enum _RSMI_IO_LINK_TYPE {
00666     RSMI_IOLINK_TYPE_UNDEFINED = 0,
00667     RSMI_IOLINK_TYPE_PCIEEXPRESS = 1,
00668     RSMI_IOLINK_TYPE_XGMI = 2,
00669     RSMI_IOLINK_TYPE_NUMIOLINKTYPES,
00670     RSMI_IOLINK_TYPE_SIZE = 0xFFFFFFFF
00671 } RSMI_IO_LINK_TYPE;
00672
00676 typedef enum {
00677     RSMI_UTILIZATION_COUNTER_FIRST = 0,
00679     RSMI_COARSE_GRAIN_GFX_ACTIVITY = RSMI_UTILIZATION_COUNTER_FIRST,
00680     RSMI_COARSE_GRAIN_MEM_ACTIVITY,
00681     RSMI_UTILIZATION_COUNTER_LAST = RSMI_COARSE_GRAIN_MEM_ACTIVITY
00682 } RSMI_UTILIZATION_COUNTER_TYPE;
00683
00687 typedef struct {
00688     RSMI_UTILIZATION_COUNTER_TYPE type;
00689     uint64_t value;
00690 } rsmi_utilization_counter_t;
00691
00695 typedef struct {
00696     uint64_t page_address;
00697     uint64_t page_size;
00698     rsmi_memory_page_status_t status;
00699 } rsmi_retired_page_record_t;
00700
00704 #define RSMI_MAX_NUM_POWER_PROFILES (sizeof(rsmi_bit_field_t) * 8)
00705
00711 typedef struct {
00715     rsmi_bit_field_t available_profiles;
00716
00720     rsmi_power_profile_preset_masks_t current;
00721
00725     uint32_t num_profiles;
00726 } rsmi_power_profile_status_t;
00728 typedef rsmi_power_profile_status_t rsmi_power_profile_status;
00730
00734 typedef struct {
00738     uint32_t num_supported;
00739
00743     uint32_t current;
00744
00749     uint64_t frequency[RSMI_MAX_NUM_FREQUENCIES];
00750 } rsmi_frequencies_t;
00752 typedef rsmi_frequencies_t rsmi_frequencies;
00754
00760 typedef struct {
00764     rsmi_frequencies_t transfer_rate;
00765
00770     uint32_t lanes[RSMI_MAX_NUM_FREQUENCIES];
00771 } rsmi_pcie_bandwidth_t;
00772
00774 typedef rsmi_pcie_bandwidth_t rsmi_pcie_bandwidth;
00776
00780 typedef struct {

```

```

00781     uint32_t major;
00782     uint32_t minor;
00783     uint32_t patch;
00784     const char *build;
00785 } rsmi_version_t;
00787 typedef rsmi_version_t rsmi_version;
00789
00792 typedef struct {
00793     uint64_t lower_bound;
00794     uint64_t upper_bound;
00795 } rsmi_range_t;
00797 typedef rsmi_range_t rsmi_range;
00799
00803 typedef struct {
00804     uint64_t frequency;
00805     uint64_t voltage;
00806 } rsmi_od_vddc_point_t;
00808 typedef rsmi_od_vddc_point_t rsmi_od_vddc_point;
00810
00816 typedef struct {
00817     rsmi_range_t freq_range;
00818     rsmi_range_t volt_range;
00819 } rsmi_freq_volt_region_t;
00821 typedef rsmi_freq_volt_region_t rsmi_freq_volt_region;
00823
00827 typedef struct {
00832     rsmi_od_vddc_point_t vc_points[RSMI_NUM_VOLTAGE_CURVE_POINTS];
00833 } rsmi_od_volt_curve_t;
00835 typedef rsmi_od_volt_curve_t rsmi_od_volt_curve;
00837
00841 typedef struct {
00842     rsmi_range_t curr_sclk_range;
00843     rsmi_range_t curr_mclk_range;
00844     rsmi_range_t sclk_freq_limits;
00845     rsmi_range_t mclk_freq_limits;
00847
00851     rsmi_od_volt_curve_t curve;
00852     uint32_t num_regions;
00853 } rsmi_od_volt_freq_data_t;
00855 typedef rsmi_od_volt_freq_data_t rsmi_od_volt_freq_data;
00857
00858
00866 struct metrics_table_header_t {
00867     // TODO(amd) Doxygen documents
00869     uint16_t structure_size;
00870     uint8_t format_revision;
00871     uint8_t content_revision;
00873 };
00874
00878 // Below is the assumed version of gpu_metric data on the device. If the device
00879 // is using this version, we can read data directly into rsmi_gpu_metrics_t.
00880 // If the device is using an older format, a conversion of formats will be
00881 // required.
00882 // DGPU targets have a format version of 1. APU targets have a format version of
00883 // 2. Currently, only version 1 (DGPU) gpu_metrics is supported.
00884 #define RSMI_GPU_METRICS_API_FORMAT_VER 1
00885 // The content version increments when gpu_metrics is extended with new and/or
00886 // existing field sizes are changed.
00887 #define RSMI_GPU_METRICS_API_CONTENT_VER_1 1
00888 #define RSMI_GPU_METRICS_API_CONTENT_VER_2 2
00889 #define RSMI_GPU_METRICS_API_CONTENT_VER_3 3
00890
00891 // This should match NUM_HBM_INSTANCES
00892 #define RSMI_NUM_HBM_INSTANCES 4
00893
00894 // Unit conversion factor for HBM temperatures
00895 #define CENTRIGRADE_TO_MILLI_CENTIGRADE 1000
00896
00897 typedef struct {
00898     // TODO(amd) Doxygen documents
00900     struct metrics_table_header_t common_header;
00901
00902     /* Temperature */
00903     uint16_t temperature_edge;
00904     uint16_t temperature_hotspot;
00905     uint16_t temperature_mem;
00906     uint16_t temperature_vrgfx;
00907     uint16_t temperature_vrsoc;
00908     uint16_t temperature_vrmem;
00909
00910     /* Utilization */
00911     uint16_t average_gfx_activity;
00912     uint16_t average_umd_activity; // memory controller
00913     uint16_t average_mm_activity; // UVD or VCN
00914
00915     /* Power/Energy */
00916     uint16_t average_socket_power;

```

```

00917     uint64_t         energy_accumulator;        // v1 mod. (32->64)
00918
00919     /* Driver attached timestamp (in ns) */
00920     uint64_t         system_clock_counter;      // v1 mod. (moved from top of struct)
00921
00922     /* Average clocks */
00923     uint16_t         average_gfxclk_frequency;
00924     uint16_t         average_socclk_frequency;
00925     uint16_t         average_uclk_frequency;
00926     uint16_t         average_vclk0_frequency;
00927     uint16_t         average_dclk0_frequency;
00928     uint16_t         average_vclk1_frequency;
00929     uint16_t         average_dclk1_frequency;
00930
00931     /* Current clocks */
00932     uint16_t         current_gfxclk;
00933     uint16_t         current_socclk;
00934     uint16_t         current_uclk;
00935     uint16_t         current_vclk0;
00936     uint16_t         current_dclk0;
00937     uint16_t         current_vclk1;
00938     uint16_t         current_dclk1;
00939
00940     /* Throttle status */
00941     uint32_t         throttle_status;
00942
00943     /* Fans */
00944     uint16_t         current_fan_speed;
00945
00946     /* Link width/speed */
00947     uint16_t         pcie_link_width;    // v1 mod.(8->16)
00948     uint16_t         pcie_link_speed;    // in 0.1 GT/s; v1 mod. (8->16)
00949
00950     uint16_t         padding;            // new in v1
00951
00952     uint32_t         gfx_activity_acc;    // new in v1
00953     uint32_t         mem_activity_acc;    // new in v1
00954     uint16_t         temperature_hbm[RSMI_NUM_HBM_INSTANCES]; // new in v1
00955 } rsmi_gpu_metrics_t;
00956
00957
00961 typedef struct {
00962     uint64_t correctable_err;
00963     uint64_t uncorrectable_err;
00964 } rsmi_error_count_t;
00965
00969 typedef struct {
00970     uint32_t process_id;
00971     uint32_t pasid;
00972     uint64_t vram_usage;
00973     uint64_t sdma_usage;
00974     uint32_t cu_occupancy;
00975 } rsmi_process_info_t;
00976
00977
00981 typedef struct rsmi_func_id_iter_handle * rsmi_func_id_iter_handle_t;
00982
00985 #define RSMI_DEFAULT_VARIANT 0xFFFFFFFFFFFFFFFF
00986
00992 typedef union id {
00993     uint64_t id;
00994     const char *name;
00995     union {
00996         rsmi_memory_type_t memory_type;
00997         rsmi_temperature_metric_t temp_metric;
01001         rsmi_event_type_t evnt_type;
01003         rsmi_event_group_t evnt_group;
01005         rsmi_clk_type_t clk_type;
01007         rsmi_fw_block_t fw_block;
01009         rsmi_gpu_block_t gpu_block_type;
01010     };
01011 } rsmi_func_id_value_t;
01012
01013
01014 /******
01032 rsmi_status_t rsmi_init(uint64_t init_flags);
01033
01039 rsmi_status_t rsmi_shut_down(void);
01040     // end of InitShut
01042
01043 /******
01060 rsmi_status_t rsmi_num_monitor_devices(uint32_t *num_devices);
01061
01089 rsmi_status_t rsmi_dev_id_get(uint32_t dv_ind, uint16_t *id);
01090
01091
01116 rsmi_status_t rsmi_dev_sku_get(uint32_t dv_ind, char *sku);
01117

```

```

01141 rsmi_status_t rsmi_dev_vendor_id_get(uint32_t dv_ind, uint16_t *id);
01142
01177 rsmi_status_t rsmi_dev_name_get(uint32_t dv_ind, char *name, size_t len);
01178
01211 rsmi_status_t rsmi_dev_brand_get(uint32_t dv_ind, char *brand, uint32_t len);
01212
01247 rsmi_status_t rsmi_dev_vendor_name_get(uint32_t dv_ind, char *name,
01248   size_t len);
01249
01272 rsmi_status_t rsmi_dev_vram_vendor_get(uint32_t dv_ind, char *brand,
01273   uint32_t len);
01274
01303 rsmi_status_t rsmi_dev_serial_number_get(uint32_t dv_ind,
01304   char *serial_num, uint32_t len);
01328 rsmi_status_t rsmi_dev_subsystem_id_get(uint32_t dv_ind, uint16_t *id);
01329
01364 rsmi_status_t
01365 rsmi_dev_subsystem_name_get(uint32_t dv_ind, char *name, size_t len);
01366
01384 rsmi_status_t
01385 rsmi_dev_drm_render_minor_get(uint32_t dv_ind, uint32_t *minor);
01386
01409 rsmi_status_t rsmi_dev_subsystem_vendor_id_get(uint32_t dv_ind, uint16_t *id);
01410
01432 rsmi_status_t rsmi_dev_unique_id_get(uint32_t dv_ind, uint64_t *id);
01433 // end of IDQuer
01435
01436 /*****
01458 rsmi_status_t
01459 rsmi_dev_pcie_bandwidth_get(uint32_t dv_ind, rsmi_pcie_bandwidth_t *bandwidth);
01460
01496 rsmi_status_t rsmi_dev_pcie_id_get(uint32_t dv_ind, uint64_t *bdfid);
01497
01520 rsmi_status_t rsmi_topo_numa_affinity_get(uint32_t dv_ind, uint32_t *numa_node);
01521
01546 rsmi_status_t rsmi_dev_pcie_throughput_get(uint32_t dv_ind, uint64_t *sent,
01547   uint64_t *received, uint64_t *max_pkt_sz);
01548
01571 rsmi_status_t rsmi_dev_pcie_replay_counter_get(uint32_t dv_ind,
01572  uint64_t *counter);
01573 // end of PCIeQuer
01575 /*****
01610 rsmi_status_t rsmi_dev_pcie_bandwidth_set(uint32_t dv_ind, uint64_t bw_bitmask);
01611 // end of PCIeCont
01613
01614 /*****
01644 rsmi_status_t
01645 rsmi_dev_power_ave_get(uint32_t dv_ind, uint32_t sensor_ind, uint64_t *power);
01646
01677 rsmi_status_t
01678 rsmi_dev_energy_count_get(uint32_t dv_ind, uint64_t *power,
01679                           float *counter_resolution, uint64_t *timestamp);
01680
01706 rsmi_status_t
01707 rsmi_dev_power_cap_get(uint32_t dv_ind, uint32_t sensor_ind, uint64_t *cap);
01708
01730 rsmi_status_t
01731 rsmi_dev_power_cap_default_get(uint32_t dv_ind, uint64_t *default_cap);
01732
01764 rsmi_status_t
01765 rsmi_dev_power_cap_range_get(uint32_t dv_ind, uint32_t sensor_ind,
01766                              uint64_t *max, uint64_t *min);
01767 // end of PowerQuer
01769
01770 /*****
01795 rsmi_status_t
01796 rsmi_dev_power_cap_set(uint32_t dv_ind, uint32_t sensor_ind, uint64_t cap);
01797
01817 rsmi_status_t
01818 rsmi_dev_power_profile_set(uint32_t dv_ind, uint32_t reserved,
01819                           rsmi_power_profile_preset_masks_t profile); // end of PowerCont
01821 /*****
01822
01823
01824
01825 /*****
01856 rsmi_status_t
01857 rsmi_dev_memory_total_get(uint32_t dv_ind, rsmi_memory_type_t mem_type,
01858                             uint64_t *total);
01859
01886 rsmi_status_t
01887 rsmi_dev_memory_usage_get(uint32_t dv_ind, rsmi_memory_type_t mem_type,
01888                            uint64_t *used);
01889
01912 rsmi_status_t
01913 rsmi_dev_memory_busy_percent_get(uint32_t dv_ind, uint32_t *busy_percent);
01914

```

```

01950 rsmi_status_t
01951 rsmi_dev_memory_reserved_pages_get(uint32_t dv_ind, uint32_t *num_pages,
01952                                     rsmi_retired_page_record_t *records); // end of MemQuer
01954
01986 rsmi_status_t rsmi_dev_fan_rpms_get(uint32_t dv_ind, uint32_t sensor_ind,
01987                                     int64_t *speed);
01988
02016 rsmi_status_t rsmi_dev_fan_speed_get(uint32_t dv_ind,
02017                                     uint32_t sensor_ind, int64_t *speed);
02018
02044 rsmi_status_t rsmi_dev_fan_speed_max_get(uint32_t dv_ind,
02045                                     uint32_t sensor_ind, uint64_t *max_speed);
02046
02077 rsmi_status_t rsmi_dev_temp_metric_get(uint32_t dv_ind, uint32_t sensor_type,
02078                                     rsmi_temperature_metric_t metric, int64_t *temperature);
02079
02110 rsmi_status_t rsmi_dev_volt_metric_get(uint32_t dv_ind,
02111                                     rsmi_voltage_type_t sensor_type,
02112                                     rsmi_voltage_metric_t metric, int64_t *voltage); // end of PhysQuer
02114
02115 /*****
02135 rsmi_status_t rsmi_dev_fan_reset(uint32_t dv_ind, uint32_t sensor_ind);
02136
02159 rsmi_status_t rsmi_dev_fan_speed_set(uint32_t dv_ind, uint32_t sensor_ind,
02160                                     uint64_t speed);
02161 // end of PhysCont
02163 *****/
02193 rsmi_status_t
02194 rsmi_dev_busy_percent_get(uint32_t dv_ind, uint32_t *busy_percent);
02195
02225 rsmi_status_t
02226 rsmi_utilization_count_get(uint32_t dv_ind,
02227                             rsmi_utilization_counter_t utilization_counters[],
02228                             uint32_t count,
02229                             uint64_t *timestamp);
02230
02254 rsmi_status_t rsmi_dev_perf_level_get(uint32_t dv_ind,
02255                                     rsmi_dev_perf_level_t *perf);
02256
02278 rsmi_status_t rsmi_perf_determinism_mode_set(uint32_t dv_ind, uint64_t clkvalue);
02279
02303 rsmi_status_t rsmi_dev_overdrive_level_get(uint32_t dv_ind, uint32_t *od);
02304
02328 rsmi_status_t rsmi_dev_mem_overdrive_level_get(uint32_t dv_ind, uint32_t *od);
02329
02360 rsmi_status_t rsmi_dev_gpu_clk_freq_get(uint32_t dv_ind,
02361                                     rsmi_clk_type_t clk_type, rsmi_frequencies_t *f);
02362
02376 rsmi_status_t rsmi_dev_gpu_reset(int32_t dv_ind);
02377
02398 rsmi_status_t rsmi_dev_od_volt_info_get(uint32_t dv_ind,
02399                                     rsmi_od_volt_freq_data_t *odv);
02400
02421 rsmi_status_t rsmi_dev_gpu_metrics_info_get(uint32_t dv_ind,
02422                                     rsmi_gpu_metrics_t *pgpu_metrics);
02423
02446 rsmi_status_t rsmi_dev_clk_range_set(uint32_t dv_ind, uint64_t minclkvalue,
02447                                     uint64_t maxclkvalue,
02448                                     rsmi_clk_type_t clkType);
02449
02472 rsmi_status_t rsmi_dev_od_clk_info_set(uint32_t dv_ind, rsmi_freq_ind_t level,
02473                                     uint64_t clkvalue,
02474                                     rsmi_clk_type_t clkType);
02475
02497 rsmi_status_t rsmi_dev_od_volt_info_set(uint32_t dv_ind, uint32_t vpoint,
02498                                     uint64_t clkvalue, uint64_t voltvalue);
02499
02538 rsmi_status_t rsmi_dev_od_volt_curve_regions_get(uint32_t dv_ind,
02539                                     uint32_t *num_regions, rsmi_freq_volt_region_t *buffer);
02540
02576 rsmi_status_t
02577 rsmi_dev_power_profile_presets_get(uint32_t dv_ind, uint32_t sensor_ind,
02578                                     rsmi_power_profile_status_t *status);
02579 // end of PerfQuer
02581 /*****
02609 rsmi_status_t
02610 rsmi_dev_perf_level_set(int32_t dv_ind, rsmi_dev_perf_level_t perf_lvl);
02611
02630 rsmi_status_t
02631 rsmi_dev_perf_level_set_v1(uint32_t dv_ind, rsmi_dev_perf_level_t perf_lvl);
02632
02676 rsmi_status_t rsmi_dev_overdrive_level_set(int32_t dv_ind, uint32_t od);
02677
02717 rsmi_status_t rsmi_dev_overdrive_level_set_v1(uint32_t dv_ind, uint32_t od);
02718
02753 rsmi_status_t rsmi_dev_gpu_clk_freq_set(uint32_t dv_ind,

```

```

02754             rsmi_clk_type_t clk_type, uint64_t freq_bitmask);
02755 // end of PerfCont
02757
02758 /*****
02777 rsmi_status_t
02778 rsmi_version_get(rsmi_version_t *version);
02779
02805 rsmi_status_t
02806 rsmi_version_str_get(rsmi_sw_component_t component, char *ver_str,
02807                     uint32_t len);
02808
02835 rsmi_status_t
02836 rsmi_dev_vbios_version_get(uint32_t dv_ind, char *vbios, uint32_t len);
02837
02862 rsmi_status_t
02863 rsmi_dev_firmware_version_get(uint32_t dv_ind, rsmi_fw_block_t block,
02864                               uint64_t *fw_version);
02865 // end of VersQuer
02867
02868 /*****
02900 rsmi_status_t rsmi_dev_ecc_count_get(uint32_t dv_ind,
02901                                     rsmi_gpu_block_t block, rsmi_error_count_t *ec);
02902
02930 rsmi_status_t rsmi_dev_ecc_enabled_get(uint32_t dv_ind,
02931                                       uint64_t *enabled_blocks);
02932
02958 rsmi_status_t rsmi_dev_ecc_status_get(uint32_t dv_ind, rsmi_gpu_block_t block,
02959                                       rsmi_ras_err_state_t *state);
02974 rsmi_status_t
02975 rsmi_status_string(rsmi_status_t status, const char **status_string);
02976 // end of ErrQuer
02978
02979 /*****
03100 rsmi_status_t
03101 rsmi_dev_counter_group_supported(uint32_t dv_ind, rsmi_event_group_t group);
03102
03131 rsmi_status_t
03132 rsmi_dev_counter_create(uint32_t dv_ind, rsmi_event_type_t type,
03133                               rsmi_event_handle_t *evnt_handle);
03134
03148 rsmi_status_t
03149 rsmi_dev_counter_destroy(rsmi_event_handle_t evnt_handle);
03150
03168 rsmi_status_t
03169 rsmi_counter_control(rsmi_event_handle_t evt_handle,
03170                     rsmi_counter_command_t cmd, void *cmd_args);
03171
03189 rsmi_status_t
03190 rsmi_counter_read(rsmi_event_handle_t evt_handle,
03191                  rsmi_counter_value_t *value);
03192
03212 rsmi_status_t
03213 rsmi_counter_available_counters_get(uint32_t dv_ind,
03214                                    rsmi_event_group_t grp, uint32_t *available); // end of PerfCntr
03216
03217 /*****
03255 rsmi_status_t
03256 rsmi_compute_process_info_get(rsmi_process_info_t *procs, uint32_t *num_items);
03257
03279 rsmi_status_t
03280 rsmi_compute_process_info_by_pid_get(uint32_t pid, rsmi_process_info_t *proc);
03281
03316 rsmi_status_t
03317 rsmi_compute_process_gpus_get(uint32_t pid, uint32_t *dv_indices,
03318                               uint32_t *num_devices);
03319 // end of SysInfo
03321
03322 /*****
03351 rsmi_status_t
03352 rsmi_dev_xgmi_error_status(uint32_t dv_ind, rsmi_xgmi_status_t *status);
03353
03366 rsmi_status_t
03367 rsmi_dev_xgmi_error_reset(uint32_t dv_ind);
03368
03387 rsmi_status_t
03388 rsmi_dev_xgmi_hive_id_get(uint32_t dv_ind, uint64_t *hive_id);
03389 // end of SysInfo
03391
03392 /*****
03415 rsmi_status_t
03416 rsmi_topo_get_numa_node_number(uint32_t dv_ind, uint32_t *numa_node);
03417
03438 rsmi_status_t
03439 rsmi_topo_get_link_weight(uint32_t dv_ind_src, uint32_t dv_ind_dst,
03440                           uint64_t *weight);
03441
03464 rsmi_status_t

```

```

03465 rsmi_minmax_bandwidth_get(uint32_t dv_ind_src, uint32_t dv_ind_dst,
03466                             uint64_t *min_bandwidth, uint64_t *max_bandwidth);
03467
03492 rsmi_status_t
03493 rsmi_topo_get_link_type(uint32_t dv_ind_src, uint32_t dv_ind_dst,
03494                          uint64_t *hops, RSMI_IO_LINK_TYPE *type);
03495
03516 rsmi_status_t
03517 rsmi_is_P2P_accessible(uint32_t dv_ind_src, uint32_t dv_ind_dst,
03518                        bool *accessible);
03519 // end of HWTopo
03521
03522 /*****
03557 rsmi_status_t
03558 rsmi_dev_compute_partition_get(uint32_t dv_ind, char *compute_partition,
03559                                uint32_t len);
03560
03583 rsmi_status_t
03584 rsmi_dev_compute_partition_set(uint32_t dv_ind,
03585                                rsmi_compute_partition_type_t compute_partition);
03586
03602 rsmi_status_t rsmi_dev_compute_partition_reset(uint32_t dv_ind);
03603 // end of ComputePartition
03605
03606 /*****
03640 rsmi_status_t
03641 rsmi_dev_nps_mode_get(uint32_t dv_ind, char *nps_mode, uint32_t len);
03642
03664 rsmi_status_t
03665 rsmi_dev_nps_mode_set(uint32_t dv_ind, rsmi_nps_mode_type_t nps_mode);
03666
03684 rsmi_status_t rsmi_dev_nps_mode_reset(uint32_t dv_ind);
03685 // end of NPSMode
03687
03688 /*****
03821 rsmi_status_t
03822 rsmi_dev_supported_func_iterator_open(uint32_t dv_ind,
03823  rsmi_func_id_iter_handle_t *handle);
03824
03849 rsmi_status_t
03850 rsmi_dev_supported_variant_iterator_open(rsmi_func_id_iter_handle_t obj_h,
03851  rsmi_func_id_iter_handle_t *var_iter);
03852
03871 rsmi_status_t
03872 rsmi_func_iter_next(rsmi_func_id_iter_handle_t handle);
03873
03885 rsmi_status_t
03886 rsmi_dev_supported_func_iterator_close(rsmi_func_id_iter_handle_t *handle);
03887
03906 rsmi_status_t
03907 rsmi_func_iter_value_get(rsmi_func_id_iter_handle_t handle,
03908                           rsmi_func_id_value_t *value);
03909 // end of APISupport
03911
03912 /*****
03932 rsmi_status_t
03933 rsmi_event_notification_init(uint32_t dv_ind);
03934
03961 rsmi_status_t
03962 rsmi_event_notification_mask_set(uint32_t dv_ind, uint64_t mask);
03963
04004 rsmi_status_t
04005 rsmi_event_notification_get(int timeout_ms,
04006                             uint32_t *num_elem, rsmi_evt_notification_data_t *data);
04007
04026 rsmi_status_t rsmi_event_notification_stop(uint32_t dv_ind);
04027 // end of EvntNotif
04029
04030 #ifdef __cplusplus
04031 }
04032 #endif // __cplusplus
04033 #endif // INCLUDE_ROCM_SMI_ROCM_SMI_H_

```

# Index

- [\\_RSMI\\_IO\\_LINK\\_TYPE](#)
  - [rocm\\_smi.h, 115](#)
- [available\\_profiles](#)
  - [rsmi\\_power\\_profile\\_status\\_t, 92](#)
- [Clock, Power and Performance Control, 49](#)
  - [rsmi\\_dev\\_gpu\\_clk\\_freq\\_set, 52](#)
  - [rsmi\\_dev\\_overdrive\\_level\\_set, 51](#)
  - [rsmi\\_dev\\_overdrive\\_level\\_set\\_v1, 51](#)
  - [rsmi\\_dev\\_perf\\_level\\_set, 50](#)
  - [rsmi\\_dev\\_perf\\_level\\_set\\_v1, 50](#)
- [Clock, Power and Performance Queries, 40](#)
  - [rsmi\\_dev\\_busy\\_percent\\_get, 41](#)
  - [rsmi\\_dev\\_clk\\_range\\_set, 46](#)
  - [rsmi\\_dev\\_gpu\\_clk\\_freq\\_get, 44](#)
  - [rsmi\\_dev\\_gpu\\_metrics\\_info\\_get, 45](#)
  - [rsmi\\_dev\\_gpu\\_reset, 44](#)
  - [rsmi\\_dev\\_mem\\_overdrive\\_level\\_get, 43](#)
  - [rsmi\\_dev\\_od\\_clk\\_info\\_set, 46](#)
  - [rsmi\\_dev\\_od\\_volt\\_curve\\_regions\\_get, 48](#)
  - [rsmi\\_dev\\_od\\_volt\\_info\\_get, 45](#)
  - [rsmi\\_dev\\_od\\_volt\\_info\\_set, 47](#)
  - [rsmi\\_dev\\_overdrive\\_level\\_get, 43](#)
  - [rsmi\\_dev\\_perf\\_level\\_get, 42](#)
  - [rsmi\\_dev\\_power\\_profile\\_presets\\_get, 48](#)
  - [rsmi\\_perf\\_determinism\\_mode\\_set, 42](#)
  - [rsmi\\_utilization\\_count\\_get, 41](#)
- [Compute Partition Functions, 71](#)
  - [rsmi\\_dev\\_compute\\_partition\\_get, 71](#)
  - [rsmi\\_dev\\_compute\\_partition\\_reset, 72](#)
  - [rsmi\\_dev\\_compute\\_partition\\_set, 72](#)
- [curr\\_mclk\\_range](#)
  - [rsmi\\_od\\_volt\\_freq\\_data\\_t, 91](#)
- [current](#)
  - [rsmi\\_frequencies\\_t, 89](#)
  - [rsmi\\_power\\_profile\\_status\\_t, 92](#)
- [Deprecated List, 5](#)
- [Error Queries, 56](#)
  - [rsmi\\_dev\\_ecc\\_count\\_get, 56](#)
  - [rsmi\\_dev\\_ecc\\_enabled\\_get, 56](#)
  - [rsmi\\_dev\\_ecc\\_status\\_get, 57](#)
  - [rsmi\\_status\\_string, 58](#)
- [Event Notification Functions, 81](#)
  - [rsmi\\_event\\_notification\\_get, 82](#)
  - [rsmi\\_event\\_notification\\_init, 81](#)
  - [rsmi\\_event\\_notification\\_mask\\_set, 81](#)
  - [rsmi\\_event\\_notification\\_stop, 83](#)
- [frequency](#)
  - [rsmi\\_frequencies\\_t, 89](#)
- [Hardware Topology Functions, 67](#)
  - [rsmi\\_is\\_P2P\\_accessible, 70](#)
  - [rsmi\\_minmax\\_bandwidth\\_get, 69](#)
  - [rsmi\\_topo\\_get\\_link\\_type, 70](#)
  - [rsmi\\_topo\\_get\\_link\\_weight, 69](#)
  - [rsmi\\_topo\\_get\\_numa\\_node\\_number, 68](#)
- [id, 85](#)
  - [memory\\_type, 86](#)
- [Identifier Queries, 14](#)
  - [rsmi\\_dev\\_brand\\_get, 17](#)
  - [rsmi\\_dev\\_drm\\_render\\_minor\\_get, 21](#)
  - [rsmi\\_dev\\_id\\_get, 15](#)
  - [rsmi\\_dev\\_name\\_get, 17](#)
  - [rsmi\\_dev\\_serial\\_number\\_get, 19](#)
  - [rsmi\\_dev\\_sku\\_get, 15](#)
  - [rsmi\\_dev\\_subsystem\\_id\\_get, 20](#)
  - [rsmi\\_dev\\_subsystem\\_name\\_get, 20](#)
  - [rsmi\\_dev\\_subsystem\\_vendor\\_id\\_get, 21](#)
  - [rsmi\\_dev\\_unique\\_id\\_get, 22](#)
  - [rsmi\\_dev\\_vendor\\_id\\_get, 16](#)
  - [rsmi\\_dev\\_vendor\\_name\\_get, 18](#)
  - [rsmi\\_dev\\_vram\\_vendor\\_get, 19](#)
  - [rsmi\\_num\\_monitor\\_devices, 15](#)
- [Initialization and Shutdown, 13](#)
  - [rsmi\\_init, 13](#)
  - [rsmi\\_shut\\_down, 14](#)
- [lanes](#)
  - [rsmi\\_pcie\\_bandwidth\\_t, 92](#)
- [Memory Queries, 32](#)
  - [rsmi\\_dev\\_memory\\_busy\\_percent\\_get, 33](#)
  - [rsmi\\_dev\\_memory\\_reserved\\_pages\\_get, 34](#)
  - [rsmi\\_dev\\_memory\\_total\\_get, 32](#)
  - [rsmi\\_dev\\_memory\\_usage\\_get, 33](#)
- [memory\\_type](#)
  - [id, 86](#)
- [metrics\\_table\\_header\\_t, 86](#)
- [NPS Mode Functions, 74](#)
  - [rsmi\\_dev\\_nps\\_mode\\_get, 74](#)
  - [rsmi\\_dev\\_nps\\_mode\\_reset, 75](#)
  - [rsmi\\_dev\\_nps\\_mode\\_set, 75](#)
- [num\\_profiles](#)
  - [rsmi\\_power\\_profile\\_status\\_t, 93](#)
- [num\\_supported](#)
  - [rsmi\\_frequencies\\_t, 89](#)

- PCIe Control, 26
  - rsmi\_dev\_pci\_bandwidth\_set, 26
- PCIe Queries, 22
  - rsmi\_dev\_pci\_bandwidth\_get, 23
  - rsmi\_dev\_pci\_id\_get, 23
  - rsmi\_dev\_pci\_replay\_counter\_get, 25
  - rsmi\_dev\_pci\_throughput\_get, 25
  - rsmi\_topo\_numa\_affinity\_get, 24
- Performance Counter Functions, 58
  - rsmi\_counter\_available\_counters\_get, 62
  - rsmi\_counter\_control, 62
  - rsmi\_counter\_read, 62
  - rsmi\_dev\_counter\_create, 60
  - rsmi\_dev\_counter\_destroy, 61
  - rsmi\_dev\_counter\_group\_supported, 60
- Physical State Control, 38
  - rsmi\_dev\_fan\_reset, 39
  - rsmi\_dev\_fan\_speed\_set, 39
- Physical State Queries, 35
  - rsmi\_dev\_fan\_rpms\_get, 35
  - rsmi\_dev\_fan\_speed\_get, 36
  - rsmi\_dev\_fan\_speed\_max\_get, 36
  - rsmi\_dev\_temp\_metric\_get, 37
  - rsmi\_dev\_volt\_metric\_get, 37
- Power Control, 30
  - rsmi\_dev\_power\_cap\_set, 31
  - rsmi\_dev\_power\_profile\_set, 31
- Power Queries, 27
  - rsmi\_dev\_energy\_count\_get, 28
  - rsmi\_dev\_power\_ave\_get, 27
  - rsmi\_dev\_power\_cap\_default\_get, 29
  - rsmi\_dev\_power\_cap\_get, 28
  - rsmi\_dev\_power\_cap\_range\_get, 29
- ROCm System Management Interface (ROCm SMI) Library, 1
- rocm\_smi.h, 97, 115
  - \_RSMI\_IO\_LINK\_TYPE, 115
  - RSMI\_CLK\_TYPE\_DCEF, 110
  - RSMI\_CLK\_TYPE\_DF, 110
  - RSMI\_CLK\_TYPE\_MEM, 110
  - RSMI\_CLK\_TYPE\_PCIE, 110
  - RSMI\_CLK\_TYPE\_SOC, 110
  - RSMI\_CLK\_TYPE\_SYS, 110
  - rsmi\_clk\_type\_t, 110
  - RSMI\_CNTR\_CMD\_START, 110
  - RSMI\_CNTR\_CMD\_STOP, 110
  - RSMI\_COARSE\_GRAIN\_MEM\_ACTIVITY, 115
  - RSMI\_COMPUTE\_PARTITION\_CPX, 110
  - RSMI\_COMPUTE\_PARTITION\_DPX, 110
  - RSMI\_COMPUTE\_PARTITION\_QPX, 110
  - RSMI\_COMPUTE\_PARTITION\_SPX, 110
  - RSMI\_COMPUTE\_PARTITION\_TPX, 110
  - rsmi\_compute\_partition\_type\_t, 110
  - rsmi\_counter\_command\_t, 109
  - RSMI\_DEFAULT\_VARIANT, 106
  - RSMI\_DEV\_PERF\_LEVEL\_AUTO, 108
  - RSMI\_DEV\_PERF\_LEVEL\_DETERMINISM, 108
  - RSMI\_DEV\_PERF\_LEVEL\_HIGH, 108
  - RSMI\_DEV\_PERF\_LEVEL\_LOW, 108
  - RSMI\_DEV\_PERF\_LEVEL\_MANUAL, 108
  - RSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_MCLK, 108
  - RSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_SCLK, 108
  - RSMI\_DEV\_PERF\_LEVEL\_STABLE\_PEAK, 108
  - RSMI\_DEV\_PERF\_LEVEL\_STABLE\_STD, 108
  - rsmi\_dev\_perf\_level\_t, 108
  - RSMI\_DEV\_PERF\_LEVEL\_UNKNOWN, 108
  - rsmi\_event\_group\_t, 108
  - rsmi\_event\_handle\_t, 107
  - RSMI\_EVENT\_MASK\_FROM\_INDEX, 106
  - rsmi\_event\_type\_t, 109
  - RSMI\_EVNT\_GRP\_XGMI, 109
  - RSMI\_EVNT\_GRP\_XGMI\_DATA\_OUT, 109
  - RSMI\_EVNT\_XGMI\_0\_BEATS\_TX, 109
  - RSMI\_EVNT\_XGMI\_0\_NOP\_TX, 109
  - RSMI\_EVNT\_XGMI\_0\_REQUEST\_TX, 109
  - RSMI\_EVNT\_XGMI\_0\_RESPONSE\_TX, 109
  - RSMI\_EVNT\_XGMI\_1\_BEATS\_TX, 109
  - RSMI\_EVNT\_XGMI\_1\_NOP\_TX, 109
  - RSMI\_EVNT\_XGMI\_1\_REQUEST\_TX, 109
  - RSMI\_EVNT\_XGMI\_1\_RESPONSE\_TX, 109
  - RSMI\_EVNT\_XGMI\_DATA\_OUT\_1, 109
  - RSMI\_EVNT\_XGMI\_DATA\_OUT\_2, 109
  - RSMI\_EVNT\_XGMI\_DATA\_OUT\_3, 109
  - RSMI\_EVNT\_XGMI\_DATA\_OUT\_4, 109
  - RSMI\_EVNT\_XGMI\_DATA\_OUT\_5, 109
  - RSMI\_EVT\_NOTIF\_VMFault, 110
  - rsmi\_evt\_notification\_type\_t, 110
  - RSMI\_FREQ\_IND\_INVALID, 114
  - RSMI\_FREQ\_IND\_MAX, 114
  - RSMI\_FREQ\_IND\_MIN, 114
  - rsmi\_freq\_ind\_t, 114
  - RSMI\_GPU\_BLOCK\_ATHUB, 113
  - RSMI\_GPU\_BLOCK\_DF, 113
  - RSMI\_GPU\_BLOCK\_FUSE, 113
  - RSMI\_GPU\_BLOCK\_GFX, 113
  - RSMI\_GPU\_BLOCK\_HDP, 113
  - RSMI\_GPU\_BLOCK\_INVALID, 113
  - RSMI\_GPU\_BLOCK\_LAST, 113
  - RSMI\_GPU\_BLOCK\_MMHUB, 113
  - RSMI\_GPU\_BLOCK\_MP0, 113
  - RSMI\_GPU\_BLOCK\_MP1, 113
  - RSMI\_GPU\_BLOCK\_PCIE\_BIF, 113
  - RSMI\_GPU\_BLOCK\_SDMA, 113
  - RSMI\_GPU\_BLOCK\_SEM, 113
  - RSMI\_GPU\_BLOCK\_SMN, 113
  - rsmi\_gpu\_block\_t, 113
  - RSMI\_GPU\_BLOCK\_UMC, 113
  - RSMI\_GPU\_BLOCK\_XGMI\_WAFL, 113
  - RSMI\_INIT\_FLAG\_ALL\_GPUS, 108
  - RSMI\_INIT\_FLAG\_RESRV\_TEST1, 108
  - rsmi\_init\_flags\_t, 108
  - RSMI\_IOLINK\_TYPE\_NUMIOLINKTYPES, 115
  - RSMI\_IOLINK\_TYPE\_PCIEEXPRESS, 115
  - RSMI\_IOLINK\_TYPE\_SIZE, 115

- RSMI\_IOLINK\_TYPE\_UNDEFINED, 115
- RSMI\_IOLINK\_TYPE\_XGMI, 115
- RSMI\_MAX\_FAN\_SPEED, 106
- RSMI\_MEM\_PAGE\_STATUS\_PENDING, 114
- RSMI\_MEM\_PAGE\_STATUS\_RESERVED, 114
- RSMI\_MEM\_PAGE\_STATUS\_UNRESERVABLE, 114
- RSMI\_MEM\_TYPE\_GTT, 114
- RSMI\_MEM\_TYPE\_VIS\_VRAM, 114
- RSMI\_MEM\_TYPE\_VRAM, 114
- rsmi\_memory\_page\_status\_t, 114
- RSMI\_MEMORY\_PARTITION\_NPS1, 111
- RSMI\_MEMORY\_PARTITION\_NPS2, 111
- RSMI\_MEMORY\_PARTITION\_NPS4, 111
- RSMI\_MEMORY\_PARTITION\_NPS8, 111
- rsmi\_memory\_type\_t, 114
- rsmi\_nps\_mode\_type\_t, 111
- rsmi\_power\_profile\_preset\_masks\_t, 112
- RSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT, 113
- RSMI\_PWR\_PROF\_PRST\_COMPUTE\_MASK, 113
- RSMI\_PWR\_PROF\_PRST\_CUSTOM\_MASK, 113
- RSMI\_PWR\_PROF\_PRST\_LAST, 113
- RSMI\_PWR\_PROF\_PRST\_POWER\_SAVING\_MASK, 113
- RSMI\_PWR\_PROF\_PRST\_VIDEO\_MASK, 113
- RSMI\_PWR\_PROF\_PRST\_VR\_MASK, 113
- RSMI\_RAS\_ERR\_STATE\_DISABLED, 114
- RSMI\_RAS\_ERR\_STATE\_ENABLED, 114
- RSMI\_RAS\_ERR\_STATE\_MULT\_UC, 114
- RSMI\_RAS\_ERR\_STATE\_NONE, 114
- RSMI\_RAS\_ERR\_STATE\_PARITY, 114
- RSMI\_RAS\_ERR\_STATE\_POISON, 114
- RSMI\_RAS\_ERR\_STATE\_SING\_C, 114
- rsmi\_ras\_err\_state\_t, 113
- RSMI\_STATUS\_AMDGPU\_RESTART\_ERR, 107
- RSMI\_STATUS\_BUSY, 107
- RSMI\_STATUS\_FILE\_ERROR, 107
- RSMI\_STATUS\_INIT\_ERROR, 107
- RSMI\_STATUS\_INPUT\_OUT\_OF\_BOUNDS, 107
- RSMI\_STATUS\_INSUFFICIENT\_SIZE, 107
- RSMI\_STATUS\_INTERNAL\_EXCEPTION, 107
- RSMI\_STATUS\_INTERRUPT, 107
- RSMI\_STATUS\_INVALID\_ARGS, 107
- RSMI\_STATUS\_NO\_DATA, 107
- RSMI\_STATUS\_NOT\_FOUND, 107
- RSMI\_STATUS\_NOT\_SUPPORTED, 107
- RSMI\_STATUS\_NOT\_YET\_IMPLEMENTED, 107
- RSMI\_STATUS\_OUT\_OF\_RESOURCES, 107
- RSMI\_STATUS\_PERMISSION, 107
- RSMI\_STATUS\_REFCOUNT\_OVERFLOW, 107
- RSMI\_STATUS\_SETTING\_UNAVAILABLE, 107
- RSMI\_STATUS\_SUCCESS, 107
- rsmi\_status\_t, 107
- RSMI\_STATUS\_UNEXPECTED\_DATA, 107
- RSMI\_STATUS\_UNEXPECTED\_SIZE, 107
- RSMI\_STATUS\_UNKNOWN\_ERROR, 107
- RSMI\_SW\_COMP\_DRIVER, 108
- rsmi\_sw\_component\_t, 108
- RSMI\_TEMP\_CRIT\_MIN, 111
- RSMI\_TEMP\_CRIT\_MIN\_HYST, 111
- RSMI\_TEMP\_CRITICAL, 111
- RSMI\_TEMP\_CRITICAL\_HYST, 111
- RSMI\_TEMP\_CURRENT, 111
- RSMI\_TEMP\_EMERGENCY, 111
- RSMI\_TEMP\_EMERGENCY\_HYST, 111
- RSMI\_TEMP\_HIGHEST, 111
- RSMI\_TEMP\_LOWEST, 111
- RSMI\_TEMP\_MAX, 111
- RSMI\_TEMP\_MAX\_HYST, 111
- RSMI\_TEMP\_MIN, 111
- RSMI\_TEMP\_MIN\_HYST, 111
- RSMI\_TEMP\_OFFSET, 111
- RSMI\_TEMP\_TYPE\_EDGE, 112
- RSMI\_TEMP\_TYPE\_HBM\_0, 112
- RSMI\_TEMP\_TYPE\_HBM\_1, 112
- RSMI\_TEMP\_TYPE\_HBM\_2, 112
- RSMI\_TEMP\_TYPE\_HBM\_3, 112
- RSMI\_TEMP\_TYPE\_INVALID, 112
- RSMI\_TEMP\_TYPE\_JUNCTION, 112
- RSMI\_TEMP\_TYPE\_MEMORY, 112
- rsmi\_temperature\_metric\_t, 111
- rsmi\_temperature\_type\_t, 112
- RSMI\_UTILIZATION\_COUNTER\_FIRST, 115
- RSMI\_UTILIZATION\_COUNTER\_TYPE, 115
- RSMI\_VOLT\_AVERAGE, 112
- RSMI\_VOLT\_CURRENT, 112
- RSMI\_VOLT\_HIGHEST, 112
- RSMI\_VOLT\_LOWEST, 112
- RSMI\_VOLT\_MAX, 112
- RSMI\_VOLT\_MAX\_CRIT, 112
- RSMI\_VOLT\_MIN, 112
- RSMI\_VOLT\_MIN\_CRIT, 112
- RSMI\_VOLT\_TYPE\_INVALID, 112
- RSMI\_VOLT\_TYPE\_VDDGFX, 112
- rsmi\_voltage\_metric\_t, 112
- rsmi\_voltage\_type\_t, 112
- RSMI\_CLK\_TYPE\_DCEF
  - rocm\_smi.h, 110
- RSMI\_CLK\_TYPE\_DF
  - rocm\_smi.h, 110
- RSMI\_CLK\_TYPE\_MEM
  - rocm\_smi.h, 110
- RSMI\_CLK\_TYPE\_PCIE
  - rocm\_smi.h, 110
- RSMI\_CLK\_TYPE\_SOC
  - rocm\_smi.h, 110
- RSMI\_CLK\_TYPE\_SYS
  - rocm\_smi.h, 110
- rsmi\_clk\_type\_t
  - rocm\_smi.h, 110
- RSMI\_CNTR\_CMD\_START
  - rocm\_smi.h, 110
- RSMI\_CNTR\_CMD\_STOP
  - rocm\_smi.h, 110

- RSMI\_COARSE\_GRAIN\_MEM\_ACTIVITY
  - rocm\_smi.h, [115](#)
- RSMI\_COMPUTE\_PARTITION\_CPX
  - rocm\_smi.h, [110](#)
- RSMI\_COMPUTE\_PARTITION\_DPX
  - rocm\_smi.h, [110](#)
- RSMI\_COMPUTE\_PARTITION\_QPX
  - rocm\_smi.h, [110](#)
- RSMI\_COMPUTE\_PARTITION\_SPX
  - rocm\_smi.h, [110](#)
- RSMI\_COMPUTE\_PARTITION\_TPX
  - rocm\_smi.h, [110](#)
- rsmi\_compute\_partition\_type\_t
  - rocm\_smi.h, [110](#)
- rsmi\_compute\_process\_gpus\_get
  - System Information Functions, [64](#)
- rsmi\_compute\_process\_info\_by\_pid\_get
  - System Information Functions, [64](#)
- rsmi\_compute\_process\_info\_get
  - System Information Functions, [63](#)
- rsmi\_counter\_available\_counters\_get
  - Performance Counter Functions, [62](#)
- rsmi\_counter\_command\_t
  - rocm\_smi.h, [109](#)
- rsmi\_counter\_control
  - Performance Counter Functions, [62](#)
- rsmi\_counter\_read
  - Performance Counter Functions, [62](#)
- rsmi\_counter\_value\_t, [86](#)
  - time\_enabled, [87](#)
  - time\_running, [87](#)
- RSMI\_DEFAULT\_VARIANT
  - rocm\_smi.h, [106](#)
- rsmi\_dev\_brand\_get
  - Identifier Queries, [17](#)
- rsmi\_dev\_busy\_percent\_get
  - Clock, Power and Performance Queries, [41](#)
- rsmi\_dev\_clk\_range\_set
  - Clock, Power and Performance Queries, [46](#)
- rsmi\_dev\_compute\_partition\_get
  - Compute Partition Functions, [71](#)
- rsmi\_dev\_compute\_partition\_reset
  - Compute Partition Functions, [72](#)
- rsmi\_dev\_compute\_partition\_set
  - Compute Partition Functions, [72](#)
- rsmi\_dev\_counter\_create
  - Performance Counter Functions, [60](#)
- rsmi\_dev\_counter\_destroy
  - Performance Counter Functions, [61](#)
- rsmi\_dev\_counter\_group\_supported
  - Performance Counter Functions, [60](#)
- rsmi\_dev\_drm\_render\_minor\_get
  - Identifier Queries, [21](#)
- rsmi\_dev\_ecc\_count\_get
  - Error Queries, [56](#)
- rsmi\_dev\_ecc\_enabled\_get
  - Error Queries, [56](#)
- rsmi\_dev\_ecc\_status\_get
  - Error Queries, [57](#)
- rsmi\_dev\_energy\_count\_get
  - Power Queries, [28](#)
- rsmi\_dev\_fan\_reset
  - Physical State Control, [39](#)
- rsmi\_dev\_fan\_rpms\_get
  - Physical State Queries, [35](#)
- rsmi\_dev\_fan\_speed\_get
  - Physical State Queries, [36](#)
- rsmi\_dev\_fan\_speed\_max\_get
  - Physical State Queries, [36](#)
- rsmi\_dev\_fan\_speed\_set
  - Physical State Control, [39](#)
- rsmi\_dev\_firmware\_version\_get
  - Version Queries, [55](#)
- rsmi\_dev\_gpu\_clk\_freq\_get
  - Clock, Power and Performance Queries, [44](#)
- rsmi\_dev\_gpu\_clk\_freq\_set
  - Clock, Power and Performance Control, [52](#)
- rsmi\_dev\_gpu\_metrics\_info\_get
  - Clock, Power and Performance Queries, [45](#)
- rsmi\_dev\_gpu\_reset
  - Clock, Power and Performance Queries, [44](#)
- rsmi\_dev\_id\_get
  - Identifier Queries, [15](#)
- rsmi\_dev\_mem\_overdrive\_level\_get
  - Clock, Power and Performance Queries, [43](#)
- rsmi\_dev\_memory\_busy\_percent\_get
  - Memory Queries, [33](#)
- rsmi\_dev\_memory\_reserved\_pages\_get
  - Memory Queries, [34](#)
- rsmi\_dev\_memory\_total\_get
  - Memory Queries, [32](#)
- rsmi\_dev\_memory\_usage\_get
  - Memory Queries, [33](#)
- rsmi\_dev\_name\_get
  - Identifier Queries, [17](#)
- rsmi\_dev\_nps\_mode\_get
  - NPS Mode Functions, [74](#)
- rsmi\_dev\_nps\_mode\_reset
  - NPS Mode Functions, [75](#)
- rsmi\_dev\_nps\_mode\_set
  - NPS Mode Functions, [75](#)
- rsmi\_dev\_od\_clk\_info\_set
  - Clock, Power and Performance Queries, [46](#)
- rsmi\_dev\_od\_volt\_curve\_regions\_get
  - Clock, Power and Performance Queries, [48](#)
- rsmi\_dev\_od\_volt\_info\_get
  - Clock, Power and Performance Queries, [45](#)
- rsmi\_dev\_od\_volt\_info\_set
  - Clock, Power and Performance Queries, [47](#)
- rsmi\_dev\_overdrive\_level\_get
  - Clock, Power and Performance Queries, [43](#)
- rsmi\_dev\_overdrive\_level\_set
  - Clock, Power and Performance Control, [51](#)
- rsmi\_dev\_overdrive\_level\_set\_v1
  - Clock, Power and Performance Control, [51](#)
- rsmi\_dev\_pci\_bandwidth\_get

- PCIe Queries, [23](#)
- `rsmi_dev_pci_bandwidth_set`
  - PCIe Control, [26](#)
- `rsmi_dev_pci_id_get`
  - PCIe Queries, [23](#)
- `rsmi_dev_pci_replay_counter_get`
  - PCIe Queries, [25](#)
- `rsmi_dev_pci_throughput_get`
  - PCIe Queries, [25](#)
- `RSMI_DEV_PERF_LEVEL_AUTO`
  - `rocm_smi.h`, [108](#)
- `RSMI_DEV_PERF_LEVEL_DETERMINISM`
  - `rocm_smi.h`, [108](#)
- `rsmi_dev_perf_level_get`
  - Clock, Power and Performance Queries, [42](#)
- `RSMI_DEV_PERF_LEVEL_HIGH`
  - `rocm_smi.h`, [108](#)
- `RSMI_DEV_PERF_LEVEL_LOW`
  - `rocm_smi.h`, [108](#)
- `RSMI_DEV_PERF_LEVEL_MANUAL`
  - `rocm_smi.h`, [108](#)
- `rsmi_dev_perf_level_set`
  - Clock, Power and Performance Control, [50](#)
- `rsmi_dev_perf_level_set_v1`
  - Clock, Power and Performance Control, [50](#)
- `RSMI_DEV_PERF_LEVEL_STABLE_MIN_MCLK`
  - `rocm_smi.h`, [108](#)
- `RSMI_DEV_PERF_LEVEL_STABLE_MIN_SCLK`
  - `rocm_smi.h`, [108](#)
- `RSMI_DEV_PERF_LEVEL_STABLE_PEAK`
  - `rocm_smi.h`, [108](#)
- `RSMI_DEV_PERF_LEVEL_STABLE_STD`
  - `rocm_smi.h`, [108](#)
- `rsmi_dev_perf_level_t`
  - `rocm_smi.h`, [108](#)
- `RSMI_DEV_PERF_LEVEL_UNKNOWN`
  - `rocm_smi.h`, [108](#)
- `rsmi_dev_power_ave_get`
  - Power Queries, [27](#)
- `rsmi_dev_power_cap_default_get`
  - Power Queries, [29](#)
- `rsmi_dev_power_cap_get`
  - Power Queries, [28](#)
- `rsmi_dev_power_cap_range_get`
  - Power Queries, [29](#)
- `rsmi_dev_power_cap_set`
  - Power Control, [31](#)
- `rsmi_dev_power_profile_presets_get`
  - Clock, Power and Performance Queries, [48](#)
- `rsmi_dev_power_profile_set`
  - Power Control, [31](#)
- `rsmi_dev_serial_number_get`
  - Identifier Queries, [19](#)
- `rsmi_dev_sku_get`
  - Identifier Queries, [15](#)
- `rsmi_dev_subsystem_id_get`
  - Identifier Queries, [20](#)
- `rsmi_dev_subsystem_name_get`
  - Identifier Queries, [20](#)
- `rsmi_dev_subsystem_vendor_id_get`
  - Identifier Queries, [21](#)
- `rsmi_dev_supported_func_iterator_close`
  - Supported Functions, [80](#)
- `rsmi_dev_supported_func_iterator_open`
  - Supported Functions, [77](#)
- `rsmi_dev_supported_variant_iterator_open`
  - Supported Functions, [79](#)
- `rsmi_dev_temp_metric_get`
  - Physical State Queries, [37](#)
- `rsmi_dev_unique_id_get`
  - Identifier Queries, [22](#)
- `rsmi_dev_vbios_version_get`
  - Version Queries, [54](#)
- `rsmi_dev_vendor_id_get`
  - Identifier Queries, [16](#)
- `rsmi_dev_vendor_name_get`
  - Identifier Queries, [18](#)
- `rsmi_dev_volt_metric_get`
  - Physical State Queries, [37](#)
- `rsmi_dev_vram_vendor_get`
  - Identifier Queries, [19](#)
- `rsmi_dev_xgmi_error_reset`
  - XGMI Functions, [66](#)
- `rsmi_dev_xgmi_error_status`
  - XGMI Functions, [66](#)
- `rsmi_dev_xgmi_hive_id_get`
  - XGMI Functions, [66](#)
- `rsmi_error_count_t`, [87](#)
- `rsmi_event_group_t`
  - `rocm_smi.h`, [108](#)
- `rsmi_event_handle_t`
  - `rocm_smi.h`, [107](#)
- `RSMI_EVENT_MASK_FROM_INDEX`
  - `rocm_smi.h`, [106](#)
- `rsmi_event_notification_get`
  - Event Notification Functions, [82](#)
- `rsmi_event_notification_init`
  - Event Notification Functions, [81](#)
- `rsmi_event_notification_mask_set`
  - Event Notification Functions, [81](#)
- `rsmi_event_notification_stop`
  - Event Notification Functions, [83](#)
- `rsmi_event_type_t`
  - `rocm_smi.h`, [109](#)
- `RSMI_EVT_GRP_XGMI`
  - `rocm_smi.h`, [109](#)
- `RSMI_EVT_GRP_XGMI_DATA_OUT`
  - `rocm_smi.h`, [109](#)
- `RSMI_EVT_XGMI_0_BEATS_TX`
  - `rocm_smi.h`, [109](#)
- `RSMI_EVT_XGMI_0_NOP_TX`
  - `rocm_smi.h`, [109](#)
- `RSMI_EVT_XGMI_0_REQUEST_TX`
  - `rocm_smi.h`, [109](#)
- `RSMI_EVT_XGMI_0_RESPONSE_TX`
  - `rocm_smi.h`, [109](#)

- RSMI\_EVNT\_XGMI\_1\_BEATS\_TX
  - rocm\_smi.h, [109](#)
- RSMI\_EVNT\_XGMI\_1\_NOP\_TX
  - rocm\_smi.h, [109](#)
- RSMI\_EVNT\_XGMI\_1\_REQUEST\_TX
  - rocm\_smi.h, [109](#)
- RSMI\_EVNT\_XGMI\_1\_RESPONSE\_TX
  - rocm\_smi.h, [109](#)
- RSMI\_EVNT\_XGMI\_DATA\_OUT\_1
  - rocm\_smi.h, [109](#)
- RSMI\_EVNT\_XGMI\_DATA\_OUT\_2
  - rocm\_smi.h, [109](#)
- RSMI\_EVNT\_XGMI\_DATA\_OUT\_3
  - rocm\_smi.h, [109](#)
- RSMI\_EVNT\_XGMI\_DATA\_OUT\_4
  - rocm\_smi.h, [109](#)
- RSMI\_EVNT\_XGMI\_DATA\_OUT\_5
  - rocm\_smi.h, [109](#)
- RSMI\_EVT\_NOTIF\_VMFAULT
  - rocm\_smi.h, [110](#)
- rsmi\_evt\_notification\_data\_t, [87](#)
- rsmi\_evt\_notification\_type\_t
  - rocm\_smi.h, [110](#)
- RSMI\_FREQ\_IND\_INVALID
  - rocm\_smi.h, [114](#)
- RSMI\_FREQ\_IND\_MAX
  - rocm\_smi.h, [114](#)
- RSMI\_FREQ\_IND\_MIN
  - rocm\_smi.h, [114](#)
- rsmi\_freq\_ind\_t
  - rocm\_smi.h, [114](#)
- rsmi\_freq\_volt\_region\_t, [88](#)
- rsmi\_frequencies\_t, [88](#)
  - current, [89](#)
  - frequency, [89](#)
  - num\_supported, [89](#)
- rsmi\_func\_iter\_next
  - Supported Functions, [79](#)
- rsmi\_func\_iter\_value\_get
  - Supported Functions, [80](#)
- RSMI\_GPU\_BLOCK\_ATHUB
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_DF
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_FUSE
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_GFX
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_HDP
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_INVALID
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_LAST
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_MMHUB
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_MPO
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_MP1
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_PCIE\_BIF
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_SDMA
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_SEM
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_SMN
  - rocm\_smi.h, [113](#)
- rsmi\_gpu\_block\_t
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_UMC
  - rocm\_smi.h, [113](#)
- RSMI\_GPU\_BLOCK\_XGMI\_WAFL
  - rocm\_smi.h, [113](#)
- rsmi\_gpu\_metrics\_t, [89](#)
- rsmi\_init
  - Initialization and Shutdown, [13](#)
- RSMI\_INIT\_FLAG\_ALL\_GPUS
  - rocm\_smi.h, [108](#)
- RSMI\_INIT\_FLAG\_RESRV\_TEST1
  - rocm\_smi.h, [108](#)
- rsmi\_init\_flags\_t
  - rocm\_smi.h, [108](#)
- RSMI\_IOLINK\_TYPE\_NUMIOLINKTYPES
  - rocm\_smi.h, [115](#)
- RSMI\_IOLINK\_TYPE\_PCIEEXPRESS
  - rocm\_smi.h, [115](#)
- RSMI\_IOLINK\_TYPE\_SIZE
  - rocm\_smi.h, [115](#)
- RSMI\_IOLINK\_TYPE\_UNDEFINED
  - rocm\_smi.h, [115](#)
- RSMI\_IOLINK\_TYPE\_XGMI
  - rocm\_smi.h, [115](#)
- rsmi\_is\_P2P\_accessible
  - Hardware Topology Functions, [70](#)
- RSMI\_MAX\_FAN\_SPEED
  - rocm\_smi.h, [106](#)
- RSMI\_MEM\_PAGE\_STATUS\_PENDING
  - rocm\_smi.h, [114](#)
- RSMI\_MEM\_PAGE\_STATUS\_RESERVED
  - rocm\_smi.h, [114](#)
- RSMI\_MEM\_PAGE\_STATUS\_UNRESERVABLE
  - rocm\_smi.h, [114](#)
- RSMI\_MEM\_TYPE\_GTT
  - rocm\_smi.h, [114](#)
- RSMI\_MEM\_TYPE\_VIS\_VRAM
  - rocm\_smi.h, [114](#)
- RSMI\_MEM\_TYPE\_VRAM
  - rocm\_smi.h, [114](#)
- rsmi\_memory\_page\_status\_t
  - rocm\_smi.h, [114](#)
- RSMI\_MEMORY\_PARTITION\_NPS1
  - rocm\_smi.h, [111](#)
- RSMI\_MEMORY\_PARTITION\_NPS2
  - rocm\_smi.h, [111](#)
- RSMI\_MEMORY\_PARTITION\_NPS4

- rocm\_smi.h, [111](#)
- RSMI\_MEMORY\_PARTITION\_NPS8
  - rocm\_smi.h, [111](#)
- rsmi\_memory\_type\_t
  - rocm\_smi.h, [114](#)
- rsmi\_minmax\_bandwidth\_get
  - Hardware Topology Functions, [69](#)
- rsmi\_nps\_mode\_type\_t
  - rocm\_smi.h, [111](#)
- rsmi\_num\_monitor\_devices
  - Identifier Queries, [15](#)
- rsmi\_od\_vddc\_point\_t, [89](#)
- rsmi\_od\_volt\_curve\_t, [90](#)
  - vc\_points, [90](#)
- rsmi\_od\_volt\_freq\_data\_t, [90](#)
  - curr\_mclk\_range, [91](#)
- rsmi\_pcie\_bandwidth\_t, [91](#)
  - lanes, [92](#)
  - transfer\_rate, [92](#)
- rsmi\_perf\_determinism\_mode\_set
  - Clock, Power and Performance Queries, [42](#)
- rsmi\_power\_profile\_preset\_masks\_t
  - rocm\_smi.h, [112](#)
- rsmi\_power\_profile\_status\_t, [92](#)
  - available\_profiles, [92](#)
  - current, [92](#)
  - num\_profiles, [93](#)
- rsmi\_process\_info\_t, [93](#)
- RSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT
  - rocm\_smi.h, [113](#)
- RSMI\_PWR\_PROF\_PRST\_COMPUTE\_MASK
  - rocm\_smi.h, [113](#)
- RSMI\_PWR\_PROF\_PRST\_CUSTOM\_MASK
  - rocm\_smi.h, [113](#)
- RSMI\_PWR\_PROF\_PRST\_LAST
  - rocm\_smi.h, [113](#)
- RSMI\_PWR\_PROF\_PRST\_POWER\_SAVING\_MASK
  - rocm\_smi.h, [113](#)
- RSMI\_PWR\_PROF\_PRST\_VIDEO\_MASK
  - rocm\_smi.h, [113](#)
- RSMI\_PWR\_PROF\_PRST\_VR\_MASK
  - rocm\_smi.h, [113](#)
- rsmi\_range\_t, [94](#)
- RSMI\_RAS\_ERR\_STATE\_DISABLED
  - rocm\_smi.h, [114](#)
- RSMI\_RAS\_ERR\_STATE\_ENABLED
  - rocm\_smi.h, [114](#)
- RSMI\_RAS\_ERR\_STATE\_MULT\_UC
  - rocm\_smi.h, [114](#)
- RSMI\_RAS\_ERR\_STATE\_NONE
  - rocm\_smi.h, [114](#)
- RSMI\_RAS\_ERR\_STATE\_PARITY
  - rocm\_smi.h, [114](#)
- RSMI\_RAS\_ERR\_STATE\_POISON
  - rocm\_smi.h, [114](#)
- RSMI\_RAS\_ERR\_STATE\_SING\_C
  - rocm\_smi.h, [114](#)
- rsmi\_ras\_err\_state\_t
  - rocm\_smi.h, [113](#)
- rsmi\_retired\_page\_record\_t, [94](#)
- rsmi\_shut\_down
  - Initialization and Shutdown, [14](#)
- RSMI\_STATUS\_AMDGPU\_RESTART\_ERR
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_BUSY
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_FILE\_ERROR
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_INIT\_ERROR
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_INPUT\_OUT\_OF\_BOUNDS
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_INSUFFICIENT\_SIZE
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_INTERNAL\_EXCEPTION
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_INTERRUPT
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_INVALID\_ARGS
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_NO\_DATA
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_NOT\_FOUND
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_NOT\_SUPPORTED
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_NOT\_YET\_IMPLEMENTED
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_OUT\_OF\_RESOURCES
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_PERMISSION
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_REFCOUNT\_OVERFLOW
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_SETTING\_UNAVAILABLE
  - rocm\_smi.h, [107](#)
- rsmi\_status\_string
  - Error Queries, [58](#)
- RSMI\_STATUS\_SUCCESS
  - rocm\_smi.h, [107](#)
- rsmi\_status\_t
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_UNEXPECTED\_DATA
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_UNEXPECTED\_SIZE
  - rocm\_smi.h, [107](#)
- RSMI\_STATUS\_UNKNOWN\_ERROR
  - rocm\_smi.h, [107](#)
- RSMI\_SW\_COMP\_DRIVER
  - rocm\_smi.h, [108](#)
- rsmi\_sw\_component\_t
  - rocm\_smi.h, [108](#)
- RSMI\_TEMP\_CRIT\_MIN
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_CRIT\_MIN\_HYST
  - rocm\_smi.h, [111](#)

- RSMI\_TEMP\_CRITICAL
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_CRITICAL\_HYST
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_CURRENT
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_EMERGENCY
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_EMERGENCY\_HYST
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_HIGHEST
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_LOWEST
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_MAX
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_MAX\_HYST
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_MIN
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_MIN\_HYST
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_OFFSET
  - rocm\_smi.h, [111](#)
- RSMI\_TEMP\_TYPE\_EDGE
  - rocm\_smi.h, [112](#)
- RSMI\_TEMP\_TYPE\_HBM\_0
  - rocm\_smi.h, [112](#)
- RSMI\_TEMP\_TYPE\_HBM\_1
  - rocm\_smi.h, [112](#)
- RSMI\_TEMP\_TYPE\_HBM\_2
  - rocm\_smi.h, [112](#)
- RSMI\_TEMP\_TYPE\_HBM\_3
  - rocm\_smi.h, [112](#)
- RSMI\_TEMP\_TYPE\_INVALID
  - rocm\_smi.h, [112](#)
- RSMI\_TEMP\_TYPE\_JUNCTION
  - rocm\_smi.h, [112](#)
- RSMI\_TEMP\_TYPE\_MEMORY
  - rocm\_smi.h, [112](#)
- rsmi\_temperature\_metric\_t
  - rocm\_smi.h, [111](#)
- rsmi\_temperature\_type\_t
  - rocm\_smi.h, [112](#)
- rsmi\_topo\_get\_link\_type
  - Hardware Topology Functions, [70](#)
- rsmi\_topo\_get\_link\_weight
  - Hardware Topology Functions, [69](#)
- rsmi\_topo\_get\_numa\_node\_number
  - Hardware Topology Functions, [68](#)
- rsmi\_topo\_numa\_affinity\_get
  - PCIe Queries, [24](#)
- rsmi\_utilization\_count\_get
  - Clock, Power and Performance Queries, [41](#)
- RSMI\_UTILIZATION\_COUNTER\_FIRST
  - rocm\_smi.h, [115](#)
- rsmi\_utilization\_counter\_t, [95](#)
- RSMI\_UTILIZATION\_COUNTER\_TYPE
  - rocm\_smi.h, [115](#)
- rsmi\_version\_get
  - Version Queries, [53](#)
- rsmi\_version\_str\_get
  - Version Queries, [54](#)
- rsmi\_version\_t, [95](#)
- RSMI\_VOLT\_AVERAGE
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_CURRENT
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_HIGHEST
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_LOWEST
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_MAX
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_MAX\_CRIT
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_MIN
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_MIN\_CRIT
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_TYPE\_INVALID
  - rocm\_smi.h, [112](#)
- RSMI\_VOLT\_TYPE\_VDDGFX
  - rocm\_smi.h, [112](#)
- rsmi\_voltage\_metric\_t
  - rocm\_smi.h, [112](#)
- rsmi\_voltage\_type\_t
  - rocm\_smi.h, [112](#)
- Supported Functions, [76](#)
  - rsmi\_dev\_supported\_func\_iterator\_close, [80](#)
  - rsmi\_dev\_supported\_func\_iterator\_open, [77](#)
  - rsmi\_dev\_supported\_variant\_iterator\_open, [79](#)
  - rsmi\_func\_iter\_next, [79](#)
  - rsmi\_func\_iter\_value\_get, [80](#)
- System Information Functions, [63](#)
  - rsmi\_compute\_process\_gpus\_get, [64](#)
  - rsmi\_compute\_process\_info\_by\_pid\_get, [64](#)
  - rsmi\_compute\_process\_info\_get, [63](#)
- time\_enabled
  - rsmi\_counter\_value\_t, [87](#)
- time\_running
  - rsmi\_counter\_value\_t, [87](#)
- transfer\_rate
  - rsmi\_pcie\_bandwidth\_t, [92](#)
- vc\_points
  - rsmi\_od\_volt\_curve\_t, [90](#)
- Version Queries, [53](#)
  - rsmi\_dev\_firmware\_version\_get, [55](#)
  - rsmi\_dev\_vbios\_version\_get, [54](#)
  - rsmi\_version\_get, [53](#)
  - rsmi\_version\_str\_get, [54](#)
- XGMI Functions, [65](#)
  - rsmi\_dev\_xgmi\_error\_reset, [66](#)

`rsmi_dev_xgmi_error_status`, [66](#)  
`rsmi_dev_xgmi_hive_id_get`, [66](#)