

AMDSMI

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>AMD System Management Interface (AMD SMI) Library</b>	<b>1</b>
<b>2</b>	<b>Deprecated List</b>	<b>5</b>
<b>3</b>	<b>Module Index</b>	<b>7</b>
3.1	Modules . . . . .	7
<b>4</b>	<b>Data Structure Index</b>	<b>9</b>
4.1	Data Structures . . . . .	9
<b>5</b>	<b>File Index</b>	<b>11</b>
5.1	File List . . . . .	11
<b>6</b>	<b>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	amdsmi_init(uint64_t init_flags) . . . . .	13
6.1.2.2	amdsmi_shut_down(void) . . . . .	14
6.2	Discovery Queries . . . . .	15
6.2.1	Detailed Description . . . . .	15
6.2.2	Function Documentation . . . . .	15
6.2.2.1	amdsmi_get_socket_handles(uint32_t *socket_count, amdsmi_socket_handle *socket_handles) . . . . .	15
6.2.2.2	amdsmi_get_socket_info(amdsmi_socket_handle socket_handle, char *name, size_t len) . . . . .	16
6.2.2.3	amdsmi_get_device_handles(amdsmi_socket_handle socket_handle, uint32_t *device_count, amdsmi_device_handle *device_handles) . . . . .	16
6.2.2.4	amdsmi_get_device_type(amdsmi_device_handle device_handle, device_type_t *device_type) . . . . .	17
6.2.2.5	amdsmi_get_device_handle_from_bdf(amdsmi_bdf_t bdf, amdsmi_device_handle *device_handle) . . . . .	17
6.3	Identifier Queries . . . . .	18
6.3.1	Detailed Description . . . . .	18
6.3.2	Function Documentation . . . . .	18
6.3.2.1	amdsmi_dev_get_id(amdsmi_device_handle device_handle, uint16_t *id) . . . . .	18
6.3.2.2	amdsmi_dev_get_vendor_name(amdsmi_device_handle device_handle, char *name, size_t len) . . . . .	19
6.3.2.3	amdsmi_dev_get_vram_vendor(amdsmi_device_handle device_handle, char *brand, uint32_t len) . . . . .	19
6.3.2.4	amdsmi_dev_get_subsystem_id(amdsmi_device_handle device_handle, uint16_t *id) . . . . .	20
6.3.2.5	amdsmi_dev_get_subsystem_name(amdsmi_device_handle device_handle, char *name, size_t len) . . . . .	20
6.3.2.6	amdsmi_dev_get_drm_render_minor(amdsmi_device_handle device_handle, uint32_t *minor) . . . . .	21

6.4	PCIe Queries	22
6.4.1	Detailed Description	22
6.4.2	Function Documentation	22
6.4.2.1	amdsmi_dev_get_pci_bandwidth(amdsmi_device_handle device_handle, amdsmi_pcie_bandwidth_t *bandwidth)	22
6.4.2.2	amdsmi_dev_get_pci_id(amdsmi_device_handle device_handle, uint64_t *bdfid)	22
6.4.2.3	amdsmi_topo_get_numa_affinity(amdsmi_device_handle device_handle, uint32_t *numa_node)	23
6.4.2.4	amdsmi_dev_get_pci_throughput(amdsmi_device_handle device_handle, uint64_t *sent, uint64_t *received, uint64_t *max_pkt_sz)	24
6.4.2.5	amdsmi_dev_get_pci_replay_counter(amdsmi_device_handle device_handle, uint64_t *counter)	24
6.5	PCIe Control	25
6.5.1	Detailed Description	25
6.5.2	Function Documentation	25
6.5.2.1	amdsmi_dev_set_pci_bandwidth(amdsmi_device_handle device_handle, uint64_t bw_bitmask)	25
6.6	Power Queries	26
6.6.1	Detailed Description	26
6.6.2	Function Documentation	26
6.6.2.1	amdsmi_dev_get_power_ave(amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t *power)	26
6.6.2.2	amdsmi_dev_get_energy_count(amdsmi_device_handle device_handle, uint64_t *power, float *counter_resolution, uint64_t *timestamp)	26
6.7	Power Control	28
6.7.1	Detailed Description	28
6.7.2	Function Documentation	28
6.7.2.1	amdsmi_dev_set_power_cap(amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t cap)	28
6.7.2.2	amdsmi_dev_set_power_profile(amdsmi_device_handle device_handle, uint32_t reserved, amdsmi_power_profile_preset_masks_t profile)	28
6.8	Memory Queries	30
6.8.1	Detailed Description	30
6.8.2	Function Documentation	30
6.8.2.1	amdsmi_dev_get_memory_total(amdsmi_device_handle device_handle, amdsmi_memory_type_t mem_type, uint64_t *total)	30
6.8.2.2	amdsmi_dev_get_memory_usage(amdsmi_device_handle device_handle, amdsmi_memory_type_t mem_type, uint64_t *used)	31
6.8.2.3	amdsmi_get_bad_page_info(amdsmi_device_handle device_handle, uint32_t *num_pages, amdsmi_retired_page_record_t *info)	31
6.8.2.4	amdsmi_get_ras_block_features_enabled(amdsmi_device_handle device_handle, amdsmi_gpu_block_t block, amdsmi_ras_err_state_t *state)	31
6.8.2.5	amdsmi_dev_get_memory_busy_percent(amdsmi_device_handle device_handle, uint32_t *busy_percent)	32
6.8.2.6	amdsmi_dev_get_memory_reserved_pages(amdsmi_device_handle device_handle, uint32_t *num_pages, amdsmi_retired_page_record_t *records)	32
6.9	Physical State Queries	34
6.9.1	Detailed Description	34
6.9.2	Function Documentation	34
6.9.2.1	amdsmi_dev_get_fan_rpms(amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t *speed)	34
6.9.2.2	amdsmi_dev_get_fan_speed(amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t *speed)	35
6.9.2.3	amdsmi_dev_get_fan_speed_max(amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t *max_speed)	35
6.9.2.4	amdsmi_dev_get_temp_metric(amdsmi_device_handle device_handle, amdsmi_temperature_type_t sensor_type, amdsmi_temperature_metric_t metric, int64_t *temperature)	35

6.9.2.5	amdsmi_dev_get_volt_metric(amdsmi_device_handle device_handle, amdsmi↔ _voltage_type_t sensor_type, amdsmi_voltage_metric_t metric, int64_t *voltage)	36
6.10	Physical State Control	37
6.10.1	Detailed Description	37
6.10.2	Function Documentation	37
6.10.2.1	amdsmi_dev_reset_fan(amdsmi_device_handle device_handle, uint32_t↔ sensor_ind)	37
6.10.2.2	amdsmi_dev_set_fan_speed(amdsmi_device_handle device_handle, uint32_t↔ sensor_ind, uint64_t speed)	37
6.11	Clock, Power and Performance Queries	39
6.11.1	Detailed Description	40
6.11.2	Function Documentation	40
6.11.2.1	amdsmi_dev_get_busy_percent(amdsmi_device_handle device_handle, uint32_t↔ *busy_percent)	40
6.11.2.2	amdsmi_get_utilization_count(amdsmi_device_handle device_handle, amdsmi↔ _utilization_counter_t utilization_counters[], uint32_t count, uint64_t *timestamp)	40
6.11.2.3	amdsmi_get_pcie_link_status(amdsmi_device_handle device_handle, amdsmi↔ _pcie_info_t *info)	41
6.11.2.4	amdsmi_get_pcie_link_caps(amdsmi_device_handle device_handle, amdsmi↔ _pcie_info_t *info)	41
6.11.2.5	amdsmi_dev_get_perf_level(amdsmi_device_handle device_handle, amdsmi↔ _dev_perf_level_t *perf)	41
6.11.2.6	amdsmi_set_perf_determinism_mode(amdsmi_device_handle device_handle, uint64_t clkvalue)	42
6.11.2.7	amdsmi_dev_get_overdrive_level(amdsmi_device_handle device_handle, uint32_t *od)	42
6.11.2.8	amdsmi_dev_get_gpu_clk_freq(amdsmi_device_handle device_handle, amdsmi↔ _clk_type_t clk_type, amdsmi_frequencies_t *f)	43
6.11.2.9	amdsmi_dev_reset_gpu(amdsmi_device_handle device_handle)	43
6.11.2.10	amdsmi_dev_get_od_volt_info(amdsmi_device_handle device_handle, amdsmi↔ _od_volt_freq_data_t *odv)	43
6.11.2.11	amdsmi_dev_get_gpu_metrics_info(amdsmi_device_handle device_handle, amdsmi_gpu_metrics_t *pgpu_metrics)	44
6.11.2.12	amdsmi_dev_set_clk_range(amdsmi_device_handle device_handle, uint64_t minclkvalue, uint64_t maxclkvalue, amdsmi_clk_type_t clkType)	44
6.11.2.13	amdsmi_dev_set_od_clk_info(amdsmi_device_handle device_handle, amdsmi↔ _freq_ind_t level, uint64_t clkvalue, amdsmi_clk_type_t clkType)	45
6.11.2.14	amdsmi_dev_set_od_volt_info(amdsmi_device_handle device_handle, uint32_t↔ vpoint, uint64_t clkvalue, uint64_t voltvalue)	45
6.11.2.15	amdsmi_dev_get_od_volt_curve_regions(amdsmi_device_handle device↔ handle, uint32_t *num_regions, amdsmi_freq_volt_region_t *buffer)	45
6.11.2.16	amdsmi_dev_get_power_profile_presets(amdsmi_device_handle device_handle, uint32_t sensor_ind, amdsmi_power_profile_status_t *status)	46
6.12	Clock, Power and Performance Control	48
6.12.1	Detailed Description	48
6.12.2	Function Documentation	48
6.12.2.1	amdsmi_dev_set_perf_level(amdsmi_device_handle device_handle, amdsmi↔ _dev_perf_level_t perf_lvl)	48
6.12.2.2	amdsmi_dev_set_perf_level_v1(amdsmi_device_handle device_handle, amdsmi↔ _dev_perf_level_t perf_lvl)	49
6.12.2.3	amdsmi_dev_set_overdrive_level(amdsmi_device_handle device_handle, uint32_t od)	49
6.12.2.4	amdsmi_dev_set_overdrive_level_v1(amdsmi_device_handle device_handle, uint32_t od)	50
6.12.2.5	amdsmi_dev_set_clk_freq(amdsmi_device_handle device_handle, amdsmi↔ _clk_type_t clk_type, uint64_t freq_bitmask)	50
6.13	Version Queries	52
6.13.1	Detailed Description	52

6.13.2	Function Documentation	52
6.13.2.1	amdsmi_get_version(amdsmi_version_t *version)	52
6.13.2.2	amdsmi_get_version_str(amdsmi_sw_component_t component, char *ver_str, uint32_t len)	52
6.14	Error Queries	54
6.14.1	Detailed Description	54
6.14.2	Function Documentation	54
6.14.2.1	amdsmi_dev_get_ecc_count(amdsmi_device_handle device_handle, amdsmi↔ gpu_block_t block, amdsmi_error_count_t *ec)	54
6.14.2.2	amdsmi_dev_get_ecc_enabled(amdsmi_device_handle device_handle, uint64↔ _t *enabled_blocks)	54
6.14.2.3	amdsmi_dev_get_ecc_status(amdsmi_device_handle device_handle, amdsmi↔ _gpu_block_t block, amdsmi_ras_err_state_t *state)	55
6.14.2.4	amdsmi_status_string(amdsmi_status_t status, const char **status_string)	55
6.15	Performance Counter Functions	57
6.15.1	Detailed Description	57
6.15.2	Function Documentation	59
6.15.2.1	amdsmi_dev_counter_group_supported(amdsmi_device_handle device_handle, amdsmi_event_group_t group)	59
6.15.2.2	amdsmi_dev_create_counter(amdsmi_device_handle device_handle, amdsmi↔ _event_type_t type, amdsmi_event_handle_t *evnt_handle)	59
6.15.2.3	amdsmi_dev_destroy_counter(amdsmi_event_handle_t evnt_handle)	60
6.15.2.4	amdsmi_control_counter(amdsmi_event_handle_t evt_handle, amdsmi↔ counter_command_t cmd, void *cmd_args)	60
6.15.2.5	amdsmi_read_counter(amdsmi_event_handle_t evt_handle, amdsmi_counter↔ value_t *value)	61
6.15.2.6	amdsmi_counter_get_available_counters(amdsmi_device_handle device↔ handle, amdsmi_event_group_t grp, uint32_t *available)	61
6.16	System Information Functions	63
6.16.1	Detailed Description	63
6.16.2	Function Documentation	63
6.16.2.1	amdsmi_get_compute_process_info(amdsmi_process_info_t *procs, uint32↔ _t *num_items)	63
6.16.2.2	amdsmi_get_compute_process_info_by_pid(uint32_t pid, amdsmi_process↔ info_t *proc)	63
6.16.2.3	amdsmi_get_compute_process_gpus(uint32_t pid, uint32_t *dv_indices, uint32_t *num_devices)	64
6.17	XGMI Functions	65
6.17.1	Detailed Description	65
6.17.2	Function Documentation	65
6.17.2.1	amdsmi_dev_xgmi_error_status(amdsmi_device_handle device_handle, amdsmi↔ _xgmi_status_t *status)	65
6.17.2.2	amdsmi_dev_reset_xgmi_error(amdsmi_device_handle device_handle)	65
6.18	Hardware Topology Functions	67
6.18.1	Detailed Description	67
6.18.2	Function Documentation	67
6.18.2.1	amdsmi_topo_get_numa_node_number(amdsmi_device_handle device_handle, uint32_t *numa_node)	67
6.18.2.2	amdsmi_topo_get_link_weight(amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, uint64_t *weight)	67
6.18.2.3	amdsmi_get_minmax_bandwidth(amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, uint64_t *min_bandwidth, uint64_t *max_bandwidth)	68
6.18.2.4	amdsmi_topo_get_link_type(amdsmi_device_handle device_handle_src, amdsmi↔ _device_handle device_handle_dst, uint64_t *hops, AMDSMI_IO_LINK_TYPE *type)	68
6.18.2.5	amdsmi_is_P2P_accessible(amdsmi_device_handle device_handle_src, amdsmi↔ _device_handle device_handle_dst, bool *accessible)	69

6.19	Supported Functions	70
6.19.1	Detailed Description	70
6.19.2	Function Documentation	71
6.19.2.1	amdsmi_dev_open_supported_func_iterator(amdsmi_device_handle device_handle, amdsmi_func_id_iter_handle_t *handle)	71
6.19.2.2	amdsmi_dev_open_supported_variant_iterator(amdsmi_func_id_iter_handle_t obj_h, amdsmi_func_id_iter_handle_t *var_iter)	72
6.19.2.3	amdsmi_next_func_iter(amdsmi_func_id_iter_handle_t handle)	72
6.19.2.4	amdsmi_dev_close_supported_func_iterator(amdsmi_func_id_iter_handle_t *handle)	73
6.19.2.5	amdsmi_get_func_iter_value(amdsmi_func_id_iter_handle_t handle, amdsmi_func_id_value_t *value)	73
6.20	Event Notification Functions	74
6.20.1	Detailed Description	74
6.20.2	Function Documentation	74
6.20.2.1	amdsmi_init_event_notification(amdsmi_device_handle device_handle)	74
6.20.2.2	amdsmi_set_event_notification_mask(amdsmi_device_handle device_handle, uint64_t mask)	74
6.20.2.3	amdsmi_get_event_notification(int timeout_ms, uint32_t *num_elem, amdsmi_evt_notification_data_t *data)	75
6.20.2.4	amdsmi_stop_event_notification(amdsmi_device_handle device_handle)	76
6.21	SW Version Information	77
6.21.1	Detailed Description	77
6.21.2	Function Documentation	77
6.21.2.1	amdsmi_get_driver_version(amdsmi_device_handle device_handle, int *length, char *version)	77
6.22	ASIC & Board Static Information	78
6.22.1	Detailed Description	78
6.22.2	Function Documentation	78
6.22.2.1	amdsmi_get_asic_info(amdsmi_device_handle device_handle, amdsmi_asic_info_t *info)	78
6.22.2.2	amdsmi_get_board_info(amdsmi_device_handle device_handle, amdsmi_board_info_t *info)	78
6.22.2.3	amdsmi_get_power_cap_info(amdsmi_device_handle device_handle, uint32_t sensor_ind, amdsmi_power_cap_info_t *info)	79
6.22.2.4	amdsmi_get_xgmi_info(amdsmi_device_handle device_handle, amdsmi_xgmi_info_t *info)	79
6.22.2.5	amdsmi_get_caps_info(amdsmi_device_handle device_handle, amdsmi_gpu_caps_t *info)	79
6.23	Firmware & VBIOS queries	81
6.23.1	Detailed Description	81
6.23.2	Function Documentation	81
6.23.2.1	amdsmi_get_fw_info(amdsmi_device_handle device_handle, amdsmi_fw_info_t *info)	81
6.23.2.2	amdsmi_get_vbios_info(amdsmi_device_handle device_handle, amdsmi_vbios_info_t *info)	81
6.24	GPU Monitoring	82
6.24.1	Detailed Description	82
6.24.2	Function Documentation	82
6.24.2.1	amdsmi_get_gpu_activity(amdsmi_device_handle device_handle, amdsmi_engine_usage_t *info)	82
6.24.2.2	amdsmi_get_power_measure(amdsmi_device_handle device_handle, amdsmi_power_measure_t *info)	82
6.24.2.3	amdsmi_get_clock_measure(amdsmi_device_handle device_handle, amdsmi_clk_type_t clk_type, amdsmi_clk_measure_t *info)	83
6.24.2.4	amdsmi_get_vram_usage(amdsmi_device_handle device_handle, amdsmi_vram_info_t *info)	83
6.25	Power Management	84

6.25.1	Detailed Description	84
6.25.2	Function Documentation	84
6.25.2.1	amdsmi_get_target_frequency_range(amdsmi_device_handle device_handle, amdsmi_clk_type_t clk_type, amdsmi_frequency_range_t *range)	84
6.26	Process information	85
6.26.1	Detailed Description	85
6.26.2	Function Documentation	85
6.26.2.1	amdsmi_get_process_list(amdsmi_device_handle device_handle, amdsmi_↔ process_handle *list, uint32_t *max_processes)	85
6.26.2.2	amdsmi_get_process_info(amdsmi_device_handle device_handle, amdsmi_↔ process_handle process, amdsmi_proc_info_t *info)	85
6.27	ECC information	87
6.27.1	Detailed Description	87
6.27.2	Function Documentation	87
6.27.2.1	amdsmi_get_ecc_error_count(amdsmi_device_handle device_handle, amdsmi_↔ _error_count_t *ec)	87
<b>7</b>	<b>Data Structure Documentation</b>	<b>89</b>
7.1	amd_metrics_table_header_t Struct Reference	89
7.1.1	Detailed Description	89
7.2	amdsmi_asic_info_t Struct Reference	89
7.2.1	Field Documentation	90
7.2.1.1	family	90
7.3	amdsmi_bdf_t Union Reference	90
7.4	amdsmi_board_info_t Struct Reference	90
7.5	amdsmi_clk_measure_t Struct Reference	91
7.6	amdsmi_counter_value_t Struct Reference	91
7.6.1	Detailed Description	91
7.6.2	Field Documentation	91
7.6.2.1	time_enabled	91
7.6.2.2	time_running	91
7.7	amdsmi_engine_usage_t Struct Reference	92
7.8	amdsmi_error_count_t Struct Reference	92
7.8.1	Detailed Description	92
7.9	amdsmi_evt_notification_data_t Struct Reference	92
7.9.1	Detailed Description	93
7.10	amdsmi_freq_volt_region_t Struct Reference	93
7.10.1	Detailed Description	93
7.11	amdsmi_frequencies_t Struct Reference	93
7.11.1	Detailed Description	94
7.11.2	Field Documentation	94
7.11.2.1	num_supported	94
7.11.2.2	current	94
7.11.2.3	frequency	94
7.12	amdsmi_frequency_range_t Struct Reference	94
7.13	amdsmi_func_id_value_t Union Reference	94
7.13.1	Detailed Description	95
7.13.2	Field Documentation	95
7.13.2.1	memory_type	95
7.14	amdsmi_fw_info_t Struct Reference	95
7.15	amdsmi_gpu_caps_t Struct Reference	96
7.15.1	Detailed Description	96
7.16	amdsmi_gpu_metrics_t Struct Reference	96
7.17	amdsmi_od_vddc_point_t Struct Reference	96
7.17.1	Detailed Description	97
7.18	amdsmi_od_volt_curve_t Struct Reference	97
7.18.1	Detailed Description	97
7.18.2	Field Documentation	97



7.18.2.1	vc_points	97
7.19	amdsmi_od_volt_freq_data_t Struct Reference	97
7.19.1	Detailed Description	98
7.19.2	Field Documentation	98
7.19.2.1	curr_mclk_range	98
7.20	amdsmi_pcie_bandwidth_t Struct Reference	98
7.20.1	Detailed Description	98
7.20.2	Field Documentation	99
7.20.2.1	transfer_rate	99
7.20.2.2	lanes	99
7.21	amdsmi_pcie_info_t Struct Reference	99
7.21.1	Detailed Description	99
7.22	amdsmi_power_cap_info_t Struct Reference	99
7.23	amdsmi_power_measure_t Struct Reference	100
7.24	amdsmi_power_profile_status_t Struct Reference	100
7.24.1	Detailed Description	100
7.24.2	Field Documentation	100
7.24.2.1	available_profiles	100
7.24.2.2	current	100
7.24.2.3	num_profiles	101
7.25	amdsmi_proc_info_t Struct Reference	101
7.25.1	Field Documentation	101
7.25.1.1	engine_usage	101
7.25.1.2	memory_usage	101
7.25.1.3	container_name	101
7.26	amdsmi_process_info_t Struct Reference	102
7.26.1	Detailed Description	102
7.27	amdsmi_range_t Struct Reference	102
7.27.1	Detailed Description	102
7.28	amdsmi_retired_page_record_t Struct Reference	103
7.28.1	Detailed Description	103
7.29	amdsmi_utilization_counter_t Struct Reference	103
7.29.1	Detailed Description	103
7.30	amdsmi_vbios_info_t Struct Reference	104
7.31	amdsmi_version_t Struct Reference	104
7.31.1	Detailed Description	104
7.32	amdsmi_vram_info_t Struct Reference	105
7.33	amdsmi_xgmi_info_t Struct Reference	105
<b>8</b>	<b>File Documentation</b>	<b>107</b>
8.1	amdsmi.h File Reference	107
8.1.1	Detailed Description	118
8.1.2	Macro Definition Documentation	118
8.1.2.1	AMDSMI_MAX_DATE_LENGTH	118
8.1.2.2	AMDSMI_MAX_FAN_SPEED	118
8.1.2.3	AMDSMI_EVENT_MASK_FROM_INDEX	118
8.1.2.4	AMDSMI_DEFAULT_VARIANT	118
8.1.3	Typedef Documentation	119
8.1.3.1	amdsmi_event_handle_t	119
8.1.4	Enumeration Type Documentation	119
8.1.4.1	amdsmi_init_flags_t	119
8.1.4.2	amdsmi_status_t	119
8.1.4.3	amdsmi_clk_type_t	120
8.1.4.4	amdsmi_dev_perf_level_t	120
8.1.4.5	amdsmi_sw_component_t	120
8.1.4.6	amdsmi_event_group_t	120
8.1.4.7	amdsmi_event_type_t	121
8.1.4.8	amdsmi_counter_command_t	121

8.1.4.9	<code>amdsmi_evt_notification_type_t</code>	121
8.1.4.10	<code>amdsmi_temperature_metric_t</code>	122
8.1.4.11	<code>amdsmi_voltage_metric_t</code>	122
8.1.4.12	<code>amdsmi_voltage_type_t</code>	123
8.1.4.13	<code>amdsmi_power_profile_preset_masks_t</code>	123
8.1.4.14	<code>amdsmi_gpu_block_t</code>	123
8.1.4.15	<code>amdsmi_ras_err_state_t</code>	124
8.1.4.16	<code>amdsmi_memory_type_t</code>	124
8.1.4.17	<code>amdsmi_freq_ind_t</code>	124
8.1.4.18	<code>amdsmi_memory_page_status_t</code>	124
8.1.4.19	<code>AMDSMI_IO_LINK_TYPE</code>	125
8.1.4.20	<code>AMDSMI_UTILIZATION_COUNTER_TYPE</code>	125
8.1.5	Function Documentation	125
8.1.5.1	<code>amdsmi_get_device_bdf(amdsmi_device_handle device_handle, amdsmi_bdf↵ _t *bdf)</code>	125
8.1.5.2	<code>amdsmi_get_device_uuid(amdsmi_device_handle device_handle, unsigned int *uuid_length, char *uuid)</code>	125

## Index

127

## Chapter 1

# AMD System Management Interface (AMD SMI) Library

The AMD System Management Interface Library, or AMD SMI library, is a C library for Linux that provides a user space interface for applications to monitor and control AMD devices.

### Supported platforms

At initial release, the AMD SMI library will support Linux bare metal and Linux virtual machine guest for AMD GPUs. In the future release, the library will be extended to support AMD EPYC™ CPUs.

AMD SMI library can run on AMD ROCm supported platforms, please refer to [List of Supported Operating Systems and GPUs](#)

To run the AMD SMI library, the amdgpu driver needs to be installed. Optionally, the libdrm can be installed to query firmware information and hardware IPs.

### Building AMD SMI

#### Additional Required software for building

In order to build the AMD 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 (v3.11.0)
- 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 AMD SMI is available on Github.

After the AMD 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,

```
1 mkdir -p build
2 cd build
3 cmake <location of root of AMD SMI library CMakeLists.txt>
4 make -j $(nproc)
5 # Install library file and header; default location is /opt/rocm
6 make instal
```

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

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

```
1 make packag
```

## 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 AMD SMI on your system and to see an example of how AMD SMI can be used, you may build and run the tests that are available in the repo. To build the tests, follow these steps:

```
1 mkdir -p build
2 cd build
3 cmake -DBUILD_TESTS=ON <location of root of AMD SMI library CMakeLists.txt>
4 make -j $(nproc)
```

## Run the Tests

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

## Usage Basics

### Device/Socket handles

Many of the functions in the library take a "socket handle" or "device handle". The socket is an abstraction of hardware physical socket. This will enable amd-smi to provide a better representation of the hardware to user. Although there is always one distinct GPU for a socket, the APU may have both GPU device and CPU device on the same socket. Moreover, for MI200, it may have multiple GCDs.

To discover the sockets in the system, `amdsmi_get_socket_handles()` is called to get list of sockets handles, which in turn can be used to query the devices in that socket using `amdsmi_get_device_handles()`. The device handler is used to distinguish the detected devices from one another. It is important to note that a device may end up with a different device handles after restart application, so a device handle should not be relied upon to be constant over process.

## Hello AMD SMI

The only required AMD-SMI call for any program that wants to use AMD-SMI is the `amdsmi_init()` call. This call initializes some internal data structures that will be used by subsequent AMD-SMI calls. In the call, a flag can be passed if the application is only interested in a specific device type.

When AMD-SMI is no longer being used, `amdsmi_shut_down()` should be called. This provides a way to do any releasing of resources that AMD-SMI may have held.

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

```
1 {c++}
2 #include <iostream>
3 #include <vector>
4 #include "amd_smi/amdsmi.h"
5
6 int main() {
7     amdsmi_status_t ret;
8
9     // Init amdsmi for sockets and devices. Here we are only interested in AMD_GPUS.
10    ret = amdsmi_init(AMD_SMI_INIT_AMD_GPUS);
11
12    // Get the socket count available in the system.
13    ret = amdsmi_get_socket_handles(&socket_count, nullptr);
14
15    // Allocate the memory for the sockets
16    std::vector<amdsmi_socket_handle> sockets(socket_count);
17    // Get the socket handles in the system
18    ret = amdsmi_get_socket_handles(&socket_count, &sockets[0]);
19
20    std::cout << "Total Socket: " << socket_count << std::endl;
21
22    // For each socket, get identifier and devices
23    for (uint32_t i=0; i < socket_count; i++) {
24        // Get Socket info
25        char socket_info[128];
26        ret = amdsmi_get_socket_info(sockets[i], socket_info, 128);
27        std::cout << "Socket " << socket_info << std::endl;
28
29        // Get the device count for the socket.
30        uint32_t device_count = 0;
31        ret = amdsmi_get_device_handles(sockets[i], &device_count, nullptr);
32
33        // Allocate the memory for the device handlers on the socket
34        std::vector<amdsmi_device_handle> device_handles(device_count);
35        // Get all devices of the socket
36        ret = amdsmi_get_device_handles(sockets[i],
37                                        &device_count, &device_handles[0]);
38
39        // For each device of the socket, get name and temperature.
40        for (uint32_t j=0; j < device_count; j++) {
41            // Get device type. Since the amdsmi is initialized with
42            // AMD_SMI_INIT_AMD_GPUS, the device_type must be AMD_GPU.
43            device_type_t device_type;
44            ret = amdsmi_get_device_type(device_handles[j], &device_type);
45            if (device_type != AMD_GPU) {
46                std::cout << "Expect AMD_GPU device type!\n";
47                return 1;
48            }
49
50            // Get device name
51            amdsmi_board_info_t board_info;
52            ret = amdsmi_get_board_info(device_handles[j], &board_info);
53            std::cout << "\tdevice "
54                    << j << "\n\t\tName: " << board_info.product_name << std::endl;
55
56            // Get temperature
57            int64_t val_i64 = 0;
58            ret = amdsmi_dev_get_temp_metric(device_handles[j], TEMPERATURE_TYPE_EDGE,
59                                            AMSMI_TEMP_CURRENT, &val_i64);
60            std::cout << "\t\tTemperature: " << val_i64/1000 << "C" << std::endl;
61        }
62    }
63
64    // Clean up resources allocated at amdsmi_init. It will invalidate sockets
65    // and devices pointers
66    ret = amdsmi_shut_down();
67
68    return 0;
69 }
```

## 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.

## Chapter 2

# Deprecated List

Global [amdsmi\\_dev\\_set\\_overdrive\\_level](#) (amdsmi\_device\_handle device\_handle, uint32\_t od)

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

Global [amdsmi\\_dev\\_set\\_perf\\_level](#) (amdsmi\_device\_handle device\_handle, amdsmi\_dev\_perf\_level\_t perf\_lvl)

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





## Chapter 3

# Module Index

### 3.1 Modules

Here is a list of all modules:

Initialization and Shutdown . . . . .	13
Discovery Queries . . . . .	15
Identifier Queries . . . . .	18
PCIe Queries . . . . .	22
PCIe Control . . . . .	25
Power Queries . . . . .	26
Power Control . . . . .	28
Memory Queries . . . . .	30
Physical State Queries . . . . .	34
Physical State Control . . . . .	37
Clock, Power and Performance Queries . . . . .	39
Clock, Power and Performance Control . . . . .	48
Version Queries . . . . .	52
Error Queries . . . . .	54
Performance Counter Functions . . . . .	57
System Information Functions . . . . .	63
XGMI Functions . . . . .	65
Hardware Topology Functions . . . . .	67
Supported Functions . . . . .	70
Event Notification Functions . . . . .	74
SW Version Information . . . . .	77
ASIC & Board Static Information . . . . .	78
Firmware & VBIOS queries . . . . .	81
GPU Monitoring . . . . .	82
Power Management . . . . .	84
Process information . . . . .	85
ECC information . . . . .	87



## Chapter 4

# Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">amd_metrics_table_header_t</a>	
The following structures hold the gpu metrics values for a device	89
<a href="#">amdsmi_asic_info_t</a>	89
<a href="#">amdsmi_bdf_t</a>	90
<a href="#">amdsmi_board_info_t</a>	90
<a href="#">amdsmi_clk_measure_t</a>	91
<a href="#">amdsmi_counter_value_t</a>	91
<a href="#">amdsmi_engine_usage_t</a>	92
<a href="#">amdsmi_error_count_t</a>	
This structure holds error counts	92
<a href="#">amdsmi_evt_notification_data_t</a>	92
<a href="#">amdsmi_freq_volt_region_t</a>	
This structure holds 2 <a href="#">amdsmi_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="#">amdsmi_od_vddc_point_t</a>	93
<a href="#">amdsmi_frequencies_t</a>	
This structure holds information about clock frequencies	93
<a href="#">amdsmi_frequency_range_t</a>	94
<a href="#">amdsmi_func_id_value_t</a>	
This union holds the value of an <a href="#">amdsmi_func_id_iter_handle_t</a> . The value may be a function name, or an enumerated variant value of types such as <a href="#">amdsmi_memory_type_t</a> , <a href="#">amdsmi_temperature_metric_t</a> , etc	94
<a href="#">amdsmi_fw_info_t</a>	95
<a href="#">amdsmi_gpu_caps_t</a>	96
<a href="#">amdsmi_gpu_metrics_t</a>	96
<a href="#">amdsmi_od_vddc_point_t</a>	
This structure represents a point on the frequency-voltage plane	96
<a href="#">amdsmi_od_volt_curve_t</a>	97
<a href="#">amdsmi_od_volt_freq_data_t</a>	
This structure holds the frequency-voltage values for a device	97
<a href="#">amdsmi_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	98
<a href="#">amdsmi_pcie_info_t</a>	
This structure holds pcie info	99
<a href="#">amdsmi_power_cap_info_t</a>	99

<a href="#">amdsmi_power_measure_t</a>	100
<a href="#">amdsmi_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	100
<a href="#">amdsmi_proc_info_t</a>	101
<a href="#">amdsmi_process_info_t</a>	
This structure contains information specific to a process	102
<a href="#">amdsmi_range_t</a>	
This structure represents a range (e.g., frequencies or voltages)	102
<a href="#">amdsmi_retired_page_record_t</a>	
Reserved Memory Page Record	103
<a href="#">amdsmi_utilization_counter_t</a>	
The utilization counter data	103
<a href="#">amdsmi_vbios_info_t</a>	104
<a href="#">amdsmi_version_t</a>	
This structure holds version information	104
<a href="#">amdsmi_vram_info_t</a>	105
<a href="#">amdsmi_xgmi_info_t</a>	105

## Chapter 5

# File Index

### 5.1 File List

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

<a href="#">amdsmi.h</a>	AMD System Management Interface API . . . . .	<a href="#">107</a>
--------------------------	---	---------------------



# Chapter 6

## Module Documentation

### 6.1 Initialization and Shutdown

#### Functions

- [amdsmi\\_status\\_t amdsmi\\_init](#) (uint64\_t init\_flags)  
*Initialize the AMD SMI library.*
- [amdsmi\\_status\\_t amdsmi\\_shut\\_down](#) (void)  
*Shutdown the AMD SMI library.*

#### 6.1.1 Detailed Description

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

#### 6.1.2 Function Documentation

##### 6.1.2.1 [amdsmi\\_status\\_t amdsmi\\_init](#) ( uint64\_t *init\_flags* )

Initialize the AMD SMI library.

This function initializes the library and the internal data structures, including those corresponding to sources of information that SMI provides.

The `init_flags` decides which type of device can be discovered by [amdsmi\\_get\\_socket\\_handles\(\)](#). `AMDSMI_INIT_AMD_GPUS` returns sockets with AMD GPUS, and `AMDSMI_INIT_AMD_GPUS | AMDSMI_INIT_AMD_CPUS` returns sockets with either AMD GPUS or CPUS. Currently, only `AMDSMI_INIT_AMD_GPUS` is supported.

#### Parameters

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

**Returns**

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.1.2.2 `amdsmi_status_t amdsmi_shut_down ( void )`**

Shutdown the AMD SMI library.

This function shuts down the library and internal data structures and performs any necessary clean ups.

**Returns**

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail



## 6.2 Discovery Queries

### Functions

- [amdsmi\\_status\\_t amdsmi\\_get\\_socket\\_handles](#) (uint32\_t \*socket\_count, amdsmi\_socket\_handle \*socket\_handles)
 

*Get the list of socket handles in the system.*
- [amdsmi\\_status\\_t amdsmi\\_get\\_socket\\_info](#) (amdsmi\_socket\_handle socket\_handle, char \*name, size\_t len)
 

*Get information about the given socket.*
- [amdsmi\\_status\\_t amdsmi\\_get\\_device\\_handles](#) (amdsmi\_socket\_handle socket\_handle, uint32\_t \*device\_count, amdsmi\_device\_handle \*device\_handles)
 

*Get the list of the device handles associated to a socket.*
- [amdsmi\\_status\\_t amdsmi\\_get\\_device\\_type](#) (amdsmi\_device\_handle device\_handle, amdsmi\_device\_type\_t \*device\_type)
 

*Get the device type of the device\_handle.*
- [amdsmi\\_status\\_t amdsmi\\_get\\_device\\_handle\\_from\\_bdf](#) (amdsmi\_bdf\_t bdf, amdsmi\_device\_handle \*device\_handle)
 

*Get device handle with the matching bdf.*

### 6.2.1 Detailed Description

These functions provide discovery of the sockets.

### 6.2.2 Function Documentation

#### 6.2.2.1 amdsmi\_status\_t amdsmi\_get\_socket\_handles ( uint32\_t \* socket\_count, amdsmi\_socket\_handle \* socket\_handles )

Get the list of socket handles in the system.

Depends on what flag is passed to [amdsmi\\_init](#). AMDSMI\_INIT\_AMD\_GPUS returns sockets with AMD GPUS, and AMDSMI\_INIT\_AMD\_GPUS | AMDSMI\_INIT\_AMD\_CPUS returns sockets with either AMD GPUS or CPUS. The socket handles can be used to query the device handles in that socket, which will be used in other APIs to get device detail information or telemtries.

#### Parameters

in, out	socket_count	As input, the value passed through this parameter is the number of ::amdsmi_socket_handle that may be safely written to the memory pointed to by socket_handles. This is the limit on how many socket handles will be written to socket_handles. On return, socket_count will contain the number of socket handles written to socket_handles, or the number of socket handles that could have been written if enough memory had been provided. If socket_handles is NULL, as output, socket_count will contain how many sockets are available to read in the system.
in, out	socket_handles	A pointer to a block of memory to which the ::amdsmi_socket_handle values will be written. This value may be NULL. In this case, this function can be used to query how many sockets are available to read in the system.

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.2.2.2 `amdsmi_status_t amdsmi_get_socket_info ( amdsmi_socket_handle socket_handle, char * name, size_t len )`

Get information about the given socket.

This function retrieves socket information. The `socket_handle` must be provided to retrieve the Socket ID.

**Parameters**

in	<code>socket_handle</code>	a socket handle
out	<code>name</code>	The id of the socket.
in	<code>len</code>	the length of the caller provided buffer <code>name</code> .

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.2.2.3 `amdsmi_status_t amdsmi_get_device_handles ( amdsmi_socket_handle socket_handle, uint32_t * device_count, amdsmi_device_handle * device_handles )`

Get the list of the device handles associated to a socket.

This function retrieves the device handles of a socket. The `socket_handle` must be provided for the device. A socket may have multiple different type devices: An APU on a socket have both CPUs and GPUs. Currently, only AMD GPUs are supported.

The number of device count is returned through `device_count` if `device_handles` is NULL. Then the number of `device_count` can be pass as input to retrieval all devices on the socket to `device_handles`.

**Parameters**

in	<code>socket_handle</code>	The socket to query
in, out	<code>device_count</code>	As input, the value passed through this parameter is the number of <a href="#">amdsmi_device_handle</a> 's that may be safely written to the memory pointed to by <code>device_handles</code> . This is the limit on how many device handles will be written to <code>device_handles</code> . On return, <code>device_count</code> will contain the number of device handles written to <code>device_handles</code> , or the number of device handles that could have been written if enough memory had been provided. If <code>device_handles</code> is NULL, as output, <code>device_count</code> will contain how many devices are available to read for the socket.
in, out	<code>device_handles</code>	A pointer to a block of memory to which the <a href="#">amdsmi_device_handle</a> values will be written. This value may be NULL. In this case, this function can be used to query how many devices are available to read.

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.2.2.4 `amdsmi_status_t amdsmi_get_device_type ( amdsmi_device_handle device_handle, device_type_t * device_type )`

Get the device type of the device\_handle.

This function retrieves the device type. A device\_handle must be provided for that device.

##### Parameters

in	<i>device_handle</i>	a device handle
out	<i>device_type</i>	a pointer to device_type_t to which the device type will be written. If this parameter is nullptr, this function will return <a href="#">AMDSMI_STATUS_INVALID</a> .

##### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.2.2.5 `amdsmi_status_t amdsmi_get_device_handle_from_bdf ( amdsmi_bdf_t bdf, amdsmi_device_handle * device_handle )`

Get device handle with the matching bdf.

Given bdf info bdf, this function will get the device handle with the matching bdf.

##### Parameters

in	<i>bdf</i>	The bdf to match with corresponding device handle.
out	<i>device_handle</i>	device handle with the matching bdf.

##### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.3 Identifier Queries

### Functions

- `amdsmi_status_t amdsmi_dev_get_id (amdsmi_device_handle device_handle, uint16_t *id)`  
*Get the device id associated with the device with provided device handler.*
- `amdsmi_status_t amdsmi_dev_get_vendor_name (amdsmi_device_handle device_handle, char *name, size_t len)`  
*Get the name string for a give vendor ID.*
- `amdsmi_status_t amdsmi_dev_get_vram_vendor (amdsmi_device_handle device_handle, char *brand, uint32_t len)`  
*Get the vram vendor string of a device.*
- `amdsmi_status_t amdsmi_dev_get_subsystem_id (amdsmi_device_handle device_handle, uint16_t *id)`  
*Get the subsystem device id associated with the device with provided device handle.*
- `amdsmi_status_t amdsmi_dev_get_subsystem_name (amdsmi_device_handle device_handle, char *name, size_t len)`  
*Get the name string for the device subsystem.*
- `amdsmi_status_t amdsmi_dev_get_drm_render_minor (amdsmi_device_handle device_handle, uint32_t *minor)`  
*Get the drm minor number associated with this device.*

### 6.3.1 Detailed Description

These functions provide identification information.

### 6.3.2 Function Documentation

#### 6.3.2.1 `amdsmi_status_t amdsmi_dev_get_id ( amdsmi_device_handle device_handle, uint16_t * id )`

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

Given a device handle `device_handle` 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. `amdsmi_dev_get_pci_id()` should be used to get a unique identifier.

#### Parameters

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

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

### 6.3.2.2 `amdsmi_status_t amsmi_dev_get_vendor_name ( amsmi_device_handle device_handle, char * name, size_t len )`

Get the name string for a give vendor ID.

Given a device handle `device_handle`, 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	<code>device_handle</code>	a device handle
in, out	<code>name</code>	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>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided, arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.
in	<code>len</code>	the length of the caller provided buffer <code>name</code> .

#### Note

`AMDSMI_STATUS_INSUFFICIENT_SIZE` is returned if `len` bytes is not large enough to hold the entire name. In this case, only `len` bytes will be written.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

### 6.3.2.3 `amdsmi_status_t amsmi_dev_get_vram_vendor ( amsmi_device_handle device_handle, char * brand, uint32_t len )`

Get the vram vendor string of a device.

This function retrieves the vram vendor name given a device handle `device_handle`, 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 `amdsmi_dev_get_vram_vendor`, then this function will return the string 'unknown' instead of the vram vendor.

#### Parameters

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

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.3.2.4 `amdsmi_status_t amdsmi_dev_get_subsystem_id ( amdsmi_device_handle device_handle, uint16_t * id )`

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

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

**Parameters**

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

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.3.2.5 `amdsmi_status_t amdsmi_dev_get_subsystem_name ( amdsmi_device_handle device_handle, char * name, size_t len )`

Get the name string for the device subsystem.

Given a device handle `device_handle`, 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>device_handle</i>	a device handle
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="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.
in	<i>len</i>	the length of the caller provided buffer <code>name</code> .

**Note**

[AMDSMI\\_STATUS\\_INSUFFICIENT\\_SIZE](#) is returned if `len` bytes is not large enough to hold the entire name. In this case, only `len` bytes will be written.

## Returns

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.3.2.6 `amdsmi_status_t amdsmi_dev_get_drm_render_minor ( amdsmi_device_handle device_handle, uint32_t * minor )`

Get the drm minor number associated with this device.

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

## Parameters

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

## Returns

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.4 PCIe Queries

### Functions

- `amdsmi_status_t amdsmi_dev_get_pcie_bandwidth (amdsmi_device_handle device_handle, amdsmi_pcie_bandwidth_t *bandwidth)`  
*Get the list of possible PCIe bandwidths that are available.*
- `amdsmi_status_t amdsmi_dev_get_pcie_id (amdsmi_device_handle device_handle, uint64_t *bdfid)`  
*Get the unique PCI device identifier associated for a device.*
- `amdsmi_status_t amdsmi_topo_get_numa_affinity (amdsmi_device_handle device_handle, uint32_t *numa_node)`  
*Get the NUMA node associated with a device.*
- `amdsmi_status_t amdsmi_dev_get_pcie_throughput (amdsmi_device_handle device_handle, uint64_t *sent, uint64_t *received, uint64_t *max_pkt_sz)`  
*Get PCIe traffic information.*
- `amdsmi_status_t amdsmi_dev_get_pcie_replay_counter (amdsmi_device_handle device_handle, uint64_t *counter)`  
*Get PCIe replay counter.*

### 6.4.1 Detailed Description

These functions provide information about PCIe.

### 6.4.2 Function Documentation

#### 6.4.2.1 `amdsmi_status_t amdsmi_dev_get_pcie_bandwidth ( amdsmi_device_handle device_handle, amdsmi_pcie_bandwidth_t * bandwidth )`

Get the list of possible PCIe bandwidths that are available.

Given a device handle `device_handle` and a pointer to a to an `amdsmi_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	<code>device_handle</code>	a device handle
in, out	<code>bandwidth</code>	a pointer to a caller provided <code>amdsmi_pcie_bandwidth_t</code> structure to which the frequency information will be written

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.4.2.2 `amdsmi_status_t amdsmi_dev_get_pcie_id ( amdsmi_device_handle device_handle, uint64_t * bdfid )`

Get the unique PCI device identifier associated for a device.



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

The format of `bdfid` will be as follows:

$$\text{BDFID} = ((\text{DOMAIN} \ \& \ 0\text{xffffffff}) \ll 32) \mid ((\text{BUS} \ \& \ 0\text{xff}) \ll 8) \mid ((\text{DEVICE} \ \& \ 0\text{x1f}) \ll 3) \mid (\text{FUNCTION} \ \& \ 0\text{x7})$$

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

#### Parameters

in	<i>device_handle</i>	a device handle
in, out	<i>bdfid</i>	a pointer to <code>uint64_t</code> to which the device bdfid value will be written. If this parameter is nullptr, this function will return <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.4.2.3 `amdsmi_status_t amdsmi_topo_get_numa_affinity ( amdsmi_device_handle device_handle, uint32_t * numa_node )`

Get the NUMA node associated with a device.

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

#### Parameters

in	<i>device_handle</i>	a device handle
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 <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

**6.4.2.4** `amdsmi_status_t amdsmi_dev_get_pci_throughput ( amdsmi_device_handle device_handle, uint64_t * sent, uint64_t * received, uint64_t * max_pkt_sz )`

Get PCIe traffic information.

Give a device handle `device_handle` 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>device_handle</i>	a device handle
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.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.4.2.5** `amdsmi_status_t amdsmi_dev_get_pci_replay_counter ( amdsmi_device_handle device_handle, uint64_t * counter )`

Get PCIe replay counter.

Given a device handle `device_handle` 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>device_handle</i>	a device handle
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="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.5 PCIe Control

### Functions

- [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_set\\_pci\\_bandwidth](#) ([amdsmi\\_device\\_handle](#) device\_handle, [uint64\\_t](#) bw\_↔ bitmask)

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

#### 6.5.1 Detailed Description

These functions provide some control over PCIe.

#### 6.5.2 Function Documentation

**6.5.2.1** [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_set\\_pci\\_bandwidth](#) ( [amdsmi\\_device\\_handle](#) device\_handle, [uint64\\_t](#) bw\_bitmask )

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

Given a device handle [device\\_handle](#) 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 [amdsmi\\_↔ frequencies\\_t](#) returned by :: [amdsmi\\_dev\\_get\\_gpu\\_clk\\_freq\(\)](#)) corresponding to that bit index will be allowed.

This function will change the performance level to [AMDSMI\\_DEV\\_PERF\\_LEVEL\\_MANUAL](#) in order to modify the set of allowable band\_widths. Caller will need to set to [AMDSMI\\_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 [amdsmi\\_frequencies\\_t::num\\_supported](#) field of [amdsmi\\_pcie\\_bandwidth\\_t](#) will be ignored.

#### Note

This function requires root access

#### Parameters

in	<a href="#">device_handle</a>	a device handle
in	<a href="#">bw_bitmask</a>	A bitmask indicating the indices of the bandwidths that are to be enabled (1) and disabled (0). Only the lowest <a href="#">amdsmi_frequencies_t::num_supported</a> (of <a href="#">amdsmi_pcie_bandwidth_t</a> ) bits of this mask are relevant.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.6 Power Queries

### Functions

- `amdsmi_status_t amdsmi_dev_get_power_ave (amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t *power)`  
*Get the average power consumption of a device.*
- `amdsmi_status_t amdsmi_dev_get_energy_count (amdsmi_device_handle device_handle, uint64_t *power, float *counter_resolution, uint64_t *timestamp)`  
*Get the energy accumulator counter of the device with provided device handle.*

### 6.6.1 Detailed Description

These functions provide information about power usage.

### 6.6.2 Function Documentation

#### 6.6.2.1 `amdsmi_status_t amdsmi_dev_get_power_ave ( amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t * power )`

Get the average power consumption of a device.

This function will write the current average power consumption (in microwatts) to the `uint64_t` pointed to by `power`, for the given device handle `device_handle` and a pointer to a `uint64_t` `power`

#### Parameters

in	<code>device_handle</code>	a device handle
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>power</code>	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 <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.6.2.2 `amdsmi_status_t amdsmi_dev_get_energy_count ( amdsmi_device_handle device_handle, uint64_t * power, float * counter_resolution, uint64_t * timestamp )`

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

Given a device handle `device_handle`, 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 `amdsmi_dev_get_power_ave()` is an average of a short time. This function accumulates all energy consumed.

## Parameters

in	<i>device_handle</i>	a device handle
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 <code>nullptr</code> , this function will return <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided, and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> 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.

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

## 6.7 Power Control

### Functions

- [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_set\\_power\\_cap](#) ([amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) sensor\_ind, [uint64\\_t](#) cap)  
*Set the maximum gpu power cap value.*
- [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_set\\_power\\_profile](#) ([amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) reserved, [amdsmi\\_power\\_profile\\_preset\\_masks\\_t](#) profile)  
*Set the power performance profile.*

### 6.7.1 Detailed Description

These functions provide ways to control power usage.

### 6.7.2 Function Documentation

**6.7.2.1** [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_set\\_power\\_cap](#) ( [amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) sensor\_ind, [uint64\\_t](#) cap )

Set the maximum gpu 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 [::amdsmi\\_dev\\_power\\_cap\\_range\\_get](#).

#### Parameters

in	<i>device_handle</i>	a device handle
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 <a href="#">uint64_t</a> that indicates the desired power cap, in microwatts

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.7.2.2** [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_set\\_power\\_profile](#) ( [amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) reserved, [amdsmi\\_power\\_profile\\_preset\\_masks\\_t](#) profile )

Set the power performance profile.

This function will attempt to set the current profile to the provided profile, given a device handle device\_handle and a profile. The provided profile must be one of the currently supported profiles, as indicated by a call to [::amdsmi\\_dev\\_get\\_power\\_profile\\_presets\(\)](#)

## Parameters

in	<i>device_handle</i>	a device handle
in	<i>reserved</i>	Not currently used. Set to 0.
in	<i>profile</i>	a <a href="#">amdsmi_power_profile_preset_masks_t</a> that hold the mask of the desired new power profile

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.8 Memory Queries

### Functions

- [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_get\\_memory\\_total](#) ([amdsmi\\_device\\_handle](#) device\_handle, [amdsmi\\_memory\\_type\\_t](#) mem\_type, [uint64\\_t](#) \*total)  
*Get the total amount of memory that exists.*
- [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_get\\_memory\\_usage](#) ([amdsmi\\_device\\_handle](#) device\_handle, [amdsmi\\_memory\\_type\\_t](#) mem\_type, [uint64\\_t](#) \*used)  
*Get the current memory usage.*
- [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_bad\\_page\\_info](#) ([amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) \*num\_pages, [amdsmi\\_retired\\_page\\_record\\_t](#) \*info)  
*The first call to this API returns the number of bad pages which should be used to allocate the buffer that should contain the bad page records.*
- [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_ras\\_block\\_features\\_enabled](#) ([amdsmi\\_device\\_handle](#) device\_handle, [amdsmi\\_gpu\\_block\\_t](#) block, [amdsmi\\_ras\\_err\\_state\\_t](#) \*state)  
*Returns if RAS features are enabled or disabled for given block.*
- [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_get\\_memory\\_busy\\_percent](#) ([amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) \*busy\_percent)  
*Get percentage of time any device memory is being used.*
- [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_get\\_memory\\_reserved\\_pages](#) ([amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) \*num\_pages, [amdsmi\\_retired\\_page\\_record\\_t](#) \*records)  
*Get information about reserved ("retired") memory pages.*

### 6.8.1 Detailed Description

These functions provide information about memory systems.

### 6.8.2 Function Documentation

**6.8.2.1** [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_get\\_memory\\_total](#) ( [amdsmi\\_device\\_handle](#) device\_handle, [amdsmi\\_memory\\_type\\_t](#) mem\_type, [uint64\\_t](#) \* total )

Get the total amount of memory that exists.

Given a device handle [device\\_handle](#), 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	<a href="#">device_handle</a>	a device handle
in	<a href="#">mem_type</a>	The type of memory for which the total amount will be found
in, out	<a href="#">total</a>	a pointer to <a href="#">uint64_t</a> to which the total amount of memory will be written If this parameter is nullptr, this function will return <a href="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided, arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.



## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.8.2.2** `amdsmi_status_t amdsmi_dev_get_memory_usage ( amdsmi_device_handle device_handle,  
amdsmi_memory_type_t mem_type, uint64_t * used )`

Get the current memory 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>device_handle</i>	a device handle
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 <a href="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.8.2.3** `amdsmi_status_t amdsmi_get_bad_page_info ( amdsmi_device_handle device_handle, uint32_t * num_pages,  
amdsmi_retired_page_record_t * info )`

The first call to this API returns the number of bad pages which should be used to allocate the buffer that should contain the bad page records.

This call will query the device `device_handle` for the number of bad pages (written to `num_pages` address). The results are written to address held by the `info` pointer.

## Parameters

in	<i>device_handle</i>	a device handle
out	<i>num_pages</i>	Number of bad page records.
out	<i>info</i>	The results will be written to the <a href="#">amdsmi_retired_page_record_t</a> pointer.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.8.2.4** `amdsmi_status_t amdsmi_get_ras_block_features_enabled ( amdsmi_device_handle device_handle,  
amdsmi_gpu_block_t block, amdsmi_ras_err_state_t * state )`

Returns if RAS features are enabled or disabled for given block.

Given a device handle `device_handle`, this function queries the state of RAS features for a specific block `block`. Result will be written to address held by pointer `state`.

#### Parameters

in	<i>device_handle</i>	Device handle which to query
in	<i>block</i>	Block which to query
in, out	<i>state</i>	A pointer to <code>amdsmi_ras_err_state_t</code> to which the state of block will be written. If this parameter is <code>nullptr</code> , this function will return <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

**6.8.2.5** `amdsmi_status_t amdsmi_dev_get_memory_busy_percent ( amdsmi_device_handle device_handle, uint32_t * busy_percent )`

Get percentage of time any device memory is being used.

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

#### Parameters

in	<i>device_handle</i>	a device handle
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>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided, arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

**6.8.2.6** `amdsmi_status_t amdsmi_dev_get_memory_reserved_pages ( amdsmi_device_handle device_handle, uint32_t * num_pages, amdsmi_retired_page_record_t * records )`

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

Given a device handle `device_handle`, this function returns retired page information `records` corresponding to the device with the provided device handle `device_handle`. 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>device_handle</i>	a device handle
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="#">amdsmi_retired_page_record_t</a> 's that may be safely written to the memory pointed to by <i>records</i> . This is the limit on how many records will be written to <i>records</i> . On return, <i>num_pages</i> will contain the number of records written to <i>records</i> , 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="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</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="#">amdsmi_retired_page_record_t</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.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.9 Physical State Queries

### Functions

- `amdsmi_status_t amdsmi_dev_get_fan_rpms (amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t *speed)`  
Get the fan speed in RPMs of the device with the specified device handle and 0-based sensor index.
- `amdsmi_status_t amdsmi_dev_get_fan_speed (amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t *speed)`  
Get the fan speed for the specified device as a value relative to [AMDSMI\\_MAX\\_FAN\\_SPEED](#).
- `amdsmi_status_t amdsmi_dev_get_fan_speed_max (amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t *max_speed)`  
Get the max. fan speed of the device with provided device handle.
- `amdsmi_status_t amdsmi_dev_get_temp_metric (amdsmi_device_handle device_handle, amdsmi_temperature_type_t sensor_type, amdsmi_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.
- `amdsmi_status_t amdsmi_dev_get_volt_metric (amdsmi_device_handle device_handle, amdsmi_voltage_type_t sensor_type, amdsmi_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.

### 6.9.1 Detailed Description

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

### 6.9.2 Function Documentation

#### 6.9.2.1 `amdsmi_status_t amdsmi_dev_get_fan_rpms ( amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t * speed )`

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

Given a device handle `device_handle` 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	<code>device_handle</code>	a device handle
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>speed</code>	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="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.9.2.2 `amdsmi_status_t amdsmi_dev_get_fan_speed ( amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t * speed )`

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

Given a device handle `device_handle` 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, [AMDSMI\\_MAX\\_FAN\\_SPEED](#)) to the `uint32_t` pointed to by `speed`

#### Parameters

in	<i>device_handle</i>	a device handle
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="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided, arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.9.2.3 `amdsmi_status_t amdsmi_dev_get_fan_speed_max ( amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t * max_speed )`

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

Given a device handle `device_handle` 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>device_handle</i>	a device handle
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 nullptr, this function will return <a href="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided, arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.9.2.4 `amdsmi_status_t amdsmi_dev_get_temp_metric ( amdsmi_device_handle device_handle, amdsmi_temperature_type_t sensor_type, amdsmi_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 handle `device_handle`, a sensor type `sensor_type`, a [amdsmi\\_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>device_handle</i>	a device handle
in	<i>sensor_type</i>	part of device from which temperature should be obtained. This should come from the enum <a href="#">amdsmi_temperature_type_t</a>
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 <a href="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided, arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.9.2.5** `amdsmi_status_t amdsmi_dev_get_volt_metric ( amdsmi_device_handle device_handle, amdsmi_voltage_type_t sensor_type, amdsmi_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 handle `device_handle`, a sensor type `sensor_type`, a [amdsmi\\_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	<i>device_handle</i>	a device handle
in	<i>sensor_type</i>	part of device from which voltage should be obtained. This should come from the enum <a href="#">amdsmi_voltage_type_t</a>
in	<i>metric</i>	enum indicated which voltage value should be retrieved
in, out	<i>voltage</i>	a pointer to <code>int64_t</code> to which the voltage will be written, in millivolts. If this parameter is <code>nullptr</code> , this function will return <a href="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided, arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.10 Physical State Control

### Functions

- [amdsmi\\_status\\_t amdsmi\\_dev\\_reset\\_fan](#) ([amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) sensor\_ind)  
*Reset the fan to automatic driver control.*
- [amdsmi\\_status\\_t amdsmi\\_dev\\_set\\_fan\\_speed](#) ([amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) sensor\_ind, [uint64\\_t](#) speed)  
*Set the fan speed for the specified device with the provided speed, in RPMs.*

### 6.10.1 Detailed Description

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

### 6.10.2 Function Documentation

#### 6.10.2.1 [amdsmi\\_status\\_t amdsmi\\_dev\\_reset\\_fan](#) ( [amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) sensor\_ind )

Reset the fan to automatic driver control.

This function returns control of the fan to the system

#### Parameters

in	<a href="#">device_handle</a>	a device handle
in	<a href="#">sensor_ind</a>	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.10.2.2 [amdsmi\\_status\\_t amdsmi\\_dev\\_set\\_fan\\_speed](#) ( [amdsmi\\_device\\_handle](#) device\_handle, [uint32\\_t](#) sensor\_ind, [uint64\\_t](#) speed )

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

Given a device handle [device\\_handle](#) 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.

#### Note

This function requires root access

**Parameters**

in	<i>device_handle</i>	a device handle
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

**Returns**

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail



## 6.11 Clock, Power and Performance Queries

### Functions

- `amdsmi_status_t amdsmi_dev_get_busy_percent (amdsmi_device_handle device_handle, uint32_t *busy_percent)`  
*Get percentage of time device is busy doing any processing.*
- `amdsmi_status_t amdsmi_get_utilization_count (amdsmi_device_handle device_handle, amdsmi_utilization_counter_t utilization_counters[], uint32_t count, uint64_t *timestamp)`  
*Get coarse grain utilization counter of the specified device.*
- `amdsmi_status_t amdsmi_get_pcie_link_status (amdsmi_device_handle device_handle, amdsmi_pcie_info_t *info)`  
*Get current PCIe info of the device with provided device handle.*
- `amdsmi_status_t amdsmi_get_pcie_link_caps (amdsmi_device_handle device_handle, amdsmi_pcie_info_t *info)`  
*Get max PCIe capabilities of the device with provided device handle.*
- `amdsmi_status_t amdsmi_dev_get_perf_level (amdsmi_device_handle device_handle, amdsmi_dev_perf_level_t *perf)`  
*Get the performance level of the device.*
- `amdsmi_status_t amdsmi_set_perf_determinism_mode (amdsmi_device_handle device_handle, uint64_t clkvalue)`  
*Enter performance determinism mode with provided device handle.*
- `amdsmi_status_t amdsmi_dev_get_overdrive_level (amdsmi_device_handle device_handle, uint32_t *od)`  
*Get the overdrive percent associated with the device with provided device handle.*
- `amdsmi_status_t amdsmi_dev_get_gpu_clk_freq (amdsmi_device_handle device_handle, amdsmi_clk_type_t clk_type, amdsmi_frequencies_t *f)`  
*Get the list of possible system clock speeds of device for a specified clock type.*
- `amdsmi_status_t amdsmi_dev_reset_gpu (amdsmi_device_handle device_handle)`  
*Reset the gpu associated with the device with provided device handle.*
- `amdsmi_status_t amdsmi_dev_get_od_volt_info (amdsmi_device_handle device_handle, amdsmi_od_volt_freq_data_t *odv)`  
*This function retrieves the voltage/frequency curve information.*
- `amdsmi_status_t amdsmi_dev_get_gpu_metrics_info (amdsmi_device_handle device_handle, amdsmi_gpu_metrics_t *pgpu_metrics)`  
*This function retrieves the gpu metrics information.*
- `amdsmi_status_t amdsmi_dev_set_clk_range (amdsmi_device_handle device_handle, uint64_t minclkvalue, uint64_t maxclkvalue, amdsmi_clk_type_t clkType)`  
*This function sets the clock range information.*
- `amdsmi_status_t amdsmi_dev_set_od_clk_info (amdsmi_device_handle device_handle, amdsmi_freq_ind_t level, uint64_t clkvalue, amdsmi_clk_type_t clkType)`  
*This function sets the clock frequency information.*
- `amdsmi_status_t amdsmi_dev_set_od_volt_info (amdsmi_device_handle device_handle, uint32_t vpoint, uint64_t clkvalue, uint64_t voltvalue)`  
*This function sets 1 of the 3 voltage curve points.*
- `amdsmi_status_t amdsmi_dev_get_od_volt_curve_regions (amdsmi_device_handle device_handle, uint32_t *num_regions, amdsmi_freq_volt_region_t *buffer)`  
*This function will retrieve the current valid regions in the frequency/voltage space.*
- `amdsmi_status_t amdsmi_dev_get_power_profile_presets (amdsmi_device_handle device_handle, uint32_t sensor_ind, amdsmi_power_profile_status_t *status)`  
*Get the list of available preset power profiles and an indication of which profile is currently active.*

### 6.11.1 Detailed Description

These functions provide information about clock frequencies and performance.

### 6.11.2 Function Documentation

#### 6.11.2.1 `amdsmi_status_t amdsmi_dev_get_busy_percent ( amdsmi_device_handle device_handle, uint32_t * busy_percent )`

Get percentage of time device is busy doing any processing.

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

##### Parameters

in	<code>device_handle</code>	a device handle
in, out	<code>busy_percent</code>	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>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

##### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.11.2.2 `amdsmi_status_t amdsmi_get_utilization_count ( amdsmi_device_handle device_handle, amdsmi_utilization_counter_t utilization_counters[], uint32_t count, uint64_t * timestamp )`

Get coarse grain utilization counter of the specified device.

Given a device handle `device_handle`, 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	<code>device_handle</code>	a device handle
in, out	<code>utilization_counters</code>	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>AMDSMI_UTILIZATION_COUNTER_TYPE</code> type for each element of the array. <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

If the function returns `AMDSMI_STATUS_SUCCESS`, the counter will be set in the value field of the `amdsmi_utilization_counter_t`.

## Parameters

in	<i>count</i>	The size of <code>utilization_counters</code> array.
----	--------------	--

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.11.2.3 `amdsmi_status_t amsmi_get_pcie_link_status ( amsmi_device_handle device_handle, amsmi_pcie_info_t * info )`

Get current PCIE info of the device with provided device handle.

Given a device handle `device_handle`, this function returns PCIE info of the given device.

## Parameters

in	<i>device_handle</i>	a device handle
out	<i>info</i>	<code>amdsmi_pcie_info_t</code> struct which will hold all the extracted PCIE info data.

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.11.2.4 `amdsmi_status_t amsmi_get_pcie_link_caps ( amsmi_device_handle device_handle, amsmi_pcie_info_t * info )`

Get max PCIe capabilities of the device with provided device handle.

Given a device handle `device_handle`, this function returns PCIe caps info of the given device.

## Parameters

in	<i>device_handle</i>	a device handle
out	<i>info</i>	<code>amdsmi_pcie_info_t</code> struct which will hold all the extracted PCIe caps data.

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.11.2.5 `amdsmi_status_t amsmi_dev_get_perf_level ( amsmi_device_handle device_handle, amsmi_dev_perf_level_t * perf )`

Get the performance level of the device.

This function will write the `amdsmi_dev_perf_level_t` to the `uint32_t` pointed to by `perf`, for a given device handle `device_handle` and a pointer to a `uint32_t` `perf`.

## Parameters

in	<i>device_handle</i>	a device handle
in, out	<i>perf</i>	a pointer to <a href="#">amdsmi_dev_perf_level_t</a> to which the performance level will be written. If this parameter is nullptr, this function will return <a href="#">AMD_SMI_STATUS_INVALID</a> if the function is supported with the provided arguments and <a href="#">AMD_SMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

## Returns

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.11.2.6 [amdsmi\\_status\\_t](#) [amdsmi\\_set\\_perf\\_determinism\\_mode](#) ( [amdsmi\\_device\\_handle](#) *device\_handle*, [uint64\\_t](#) *clkvalue* )

Enter performance determinism mode with provided device handle.

Given a device handle *device\_handle* 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 [amdsmi\\_dev\\_perf\\_level\\_t](#) of the device being [AMD\\_SMI\\_DEV\\_PERF\\_LEVEL\\_DETERMINISM](#).

## Parameters

in	<i>device_handle</i>	a device handle
in	<i>clkvalue</i>	Softmax value for GFXCLK in MHz.

## Returns

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.11.2.7 [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_get\\_overdrive\\_level](#) ( [amdsmi\\_device\\_handle](#) *device\_handle*, [uint32\\_t](#) \* *od* )

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

Given a device handle *device\_handle* 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>device_handle</i>	a device handle
in, out	<i>od</i>	a pointer to <a href="#">uint32_t</a> to which the overdrive percentage will be written. If this parameter is nullptr, this function will return <a href="#">AMD_SMI_STATUS_INVALID</a> if the function is supported with the provided arguments and <a href="#">AMD_SMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.11.2.8** `amdsmi_status_t amdsmi_dev_get_gpu_clk_freq ( amdsmi_device_handle device_handle,  
amdsmi_clk_type_t clk_type, amdsmi_frequencies_t * f )`

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

Given a device handle `device_handle`, a clock type `clk_type`, and a pointer to a to an [amdsmi\\_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	<code>device_handle</code>	a device handle
in	<code>clk_type</code>	the type of clock for which the frequency is desired
in, out	<code>f</code>	a pointer to a caller provided <a href="#">amdsmi_frequencies_t</a> structure to which the frequency information will be written. Frequency values are in Hz.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.11.2.9** `amdsmi_status_t amdsmi_dev_reset_gpu ( amdsmi_device_handle device_handle )`

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

Given a device handle `device_handle`, this function will reset the GPU

## Parameters

in	<code>device_handle</code>	a device handle
----	----------------------------	-----------------

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.11.2.10** `amdsmi_status_t amdsmi_dev_get_od_volt_info ( amdsmi_device_handle device_handle,  
amdsmi_od_volt_freq_data_t * odv )`

This function retrieves the voltage/frequency curve information.

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

## Parameters

in	<i>device_handle</i>	a device handle
in, out	<i>odv</i>	a pointer to an <a href="#">amdsmi_od_volt_freq_data_t</a> structure If this parameter is nullptr, this function will return <a href="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided, arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

6.11.2.11 **amdsmi\_status\_t** **amdsmi\_dev\_get\_gpu\_metrics\_info** ( **amdsmi\_device\_handle** *device\_handle*, **amdsmi\_gpu\_metrics\_t** \* *pgpu\_metrics* )

This function retrieves the gpu metrics information.

Given a device handle *device\_handle* and a pointer to a [amdsmi\\_gpu\\_metrics\\_t](#) structure *pgpu\_metrics*, this function will populate *pgpu\_metrics*. See [amdsmi\\_gpu\\_metrics\\_t](#) for more details.

## Parameters

in	<i>device_handle</i>	a device handle
in, out	<i>pgpu_metrics</i>	a pointer to an <a href="#">amdsmi_gpu_metrics_t</a> structure If this parameter is nullptr, this function will return <a href="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided, arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

6.11.2.12 **amdsmi\_status\_t** **amdsmi\_dev\_set\_clk\_range** ( **amdsmi\_device\_handle** *device\_handle*, **uint64\_t** *minclkvalue*, **uint64\_t** *maxclkvalue*, **amdsmi\_clk\_type\_t** *clkType* )

This function sets the clock range information.

Given a device handle *device\_handle*, 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>device_handle</i>	a device handle
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>	<a href="#">AMDSMI_CLK_TYPE_SYS</a>   <a href="#">AMDSMI_CLK_TYPE_MEM</a> range type

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

6.11.2.13 `amdsmi_status_t amsmi_dev_set_od_clk_info ( amsmi_device_handle device_handle, amsmi_freq_ind_t level, uint64_t clkvalue, amsmi_clk_type_t clkType )`

This function sets the clock frequency information.

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

## Parameters

in	<code>device_handle</code>	a device handle
in	<code>level</code>	AMDSMI_FREQ_IND_MIN AMDSMI_FREQ_IND_MAX to set the minimum (0) or maximum (1) speed.
in	<code>clkvalue</code>	value to apply to the clock range. Frequency values are in MHz.
in	<code>clkType</code>	AMDSMI_CLK_TYPE_SYS   AMDSMI_CLK_TYPE_MEM range type

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

6.11.2.14 `amdsmi_status_t amsmi_dev_set_od_volt_info ( amsmi_device_handle device_handle, uint32_t vpoint, uint64_t clkvalue, uint64_t voltvalue )`

This function sets 1 of the 3 voltage curve points.

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

## Parameters

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

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

6.11.2.15 `amdsmi_status_t amsmi_dev_get_od_volt_curve_regions ( amsmi_device_handle device_handle, uint32_t * num_regions, amsmi_freq_volt_region_t * buffer )`

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

Given a device handle `device_handle`, a pointer to an unsigned integer `num_regions` and a buffer of `amdsmi_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 `amdsmi_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 :: `amdsmi_dev_get_od_volt_info()`.

#### Parameters

in	<i>device_handle</i>	a device handle
in, out	<i>num_regions</i>	As input, this is the number of <code>amdsmi_freq_volt_region_t</code> structures that can be written to <code>buffer</code> . As output, this is the number of <code>amdsmi_freq_volt_region_t</code> structures that were actually written. If this parameter is nullptr, this function will return <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided, arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.
in, out	<i>buffer</i>	a caller provided buffer to which <code>amdsmi_freq_volt_region_t</code> structures will be written. If this parameter is nullptr, this function will return <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided, arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

**6.11.2.16** `amdsmi_status_t amdsmi_dev_get_power_profile_presets ( amdsmi_device_handle device_handle, uint32_t sensor_ind, amdsmi_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 handle `device_handle` and a pointer to a `amdsmi_power_profile_status_t` `status`, this function will set the bits of the `amdsmi_power_profile_status_t.available_profiles` bit field of `status` to 1 if the profile corresponding to the respective `amdsmi_power_profile_preset_masks_t` profiles are enabled. For example, if both the VIDEO and VR power profiles are available selections, then `AMDSMI_PWR_PROF_PRST_VIDEO_MASK` AND'ed with `amdsmi_power_profile_status_t.available_profiles` will be non-zero as will `AMDSMI_PWR_PROF_PRST_VR_MASK` AND'ed with `amdsmi_power_profile_status_t.available_profiles`. Additionally, `amdsmi_power_profile_status_t.current` will be set to the `amdsmi_power_profile_preset_masks_t` of the profile that is currently active.

#### Parameters

in	<i>device_handle</i>	a device handle
in	<i>sensor_ind</i>	a 0-based sensor index. Normally, this will be 0. If a device has more than one sensor, it could be greater than 0.
in, out	<i>status</i>	a pointer to <code>amdsmi_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>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided, arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.



**Returns**

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

## 6.12 Clock, Power and Performance Control

### Functions

- `amdsmi_status_t amdsmi_dev_set_perf_level (amdsmi_device_handle device_handle, amdsmi_dev_perf_level_t perf_lvl)`  
*Set the PowerPlay performance level associated with the device with provided device handle with the provided value.*
- `amdsmi_status_t amdsmi_dev_set_perf_level_v1 (amdsmi_device_handle device_handle, amdsmi_dev_perf_level_t perf_lvl)`  
*Set the PowerPlay performance level associated with the device with provided device handle with the provided value.*
- `amdsmi_status_t amdsmi_dev_set_overdrive_level (amdsmi_device_handle device_handle, uint32_t od)`  
*Set the overdrive percent associated with the device with provided device handle with the provided value. See details for WARNING.*
- `amdsmi_status_t amdsmi_dev_set_overdrive_level_v1 (amdsmi_device_handle device_handle, uint32_t od)`  
*Set the overdrive percent associated with the device with provided device handle with the provided value. See details for WARNING.*
- `amdsmi_status_t amdsmi_dev_set_clk_freq (amdsmi_device_handle device_handle, amdsmi_clk_type_t clk_type, uint64_t freq_bitmask)`  
*Control the set of allowed frequencies that can be used for the specified clock.*

### 6.12.1 Detailed Description

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

### 6.12.2 Function Documentation

#### 6.12.2.1 `amdsmi_status_t amdsmi_dev_set_perf_level ( amdsmi_device_handle device_handle, amdsmi_dev_perf_level_t perf_lvl )`

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

**Deprecated** :: `amdsmi_dev_set_perf_level_v1()` is preferred, with an interface that more closely matches the rest of the `amd_smi` API.

Given a device handle `device_handle` and an `amdsmi_dev_perf_level_t perf_level`, this function will set the PowerPlay performance level for the device to the value `perf_lvl`.

#### Note

This function requires root access

#### Parameters

in	<code>device_handle</code>	a device handle
in	<code>perf_lvl</code>	the value to which the performance level should be set

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

### 6.12.2.2 `amdsmi_status_t amsmi_dev_set_perf_level_v1 ( amsmi_device_handle device_handle, amsmi_dev_perf_level_t perf_lvl )`

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

Given a device handle `device_handle` and an `amdsmi_dev_perf_level_t` `perf_level`, this function will set the PowerPlay performance level for the device to the value `perf_lvl`.

## Note

This function requires root access

## Parameters

in	<code>device_handle</code>	a device handle
in	<code>perf_lvl</code>	the value to which the performance level should be set

## Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

### 6.12.2.3 `amdsmi_status_t amsmi_dev_set_overdrive_level ( amsmi_device_handle device_handle, uint32_t od )`

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

**Deprecated** This function is deprecated. :: `amdsmi_dev_set_overdrive_level_v1` has the same functionality, with an interface that more closely matches the rest of the `amd_smi` API.

Given a device handle `device_handle` 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>device_handle</i>	a device handle
in	<i>od</i>	the value to which the overdrive level should be set

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.12.2.4 `amdsmi_status_t amdsmi_dev_set_overdrive_level_v1 ( amdsmi_device_handle device_handle, uint32_t od )`

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

Given a device handle `device_handle` 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.

**Note**

This function requires root access

**Parameters**

in	<i>device_handle</i>	a device handle
in	<i>od</i>	the value to which the overdrive level should be set

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.12.2.5 `amdsmi_status_t amdsmi_dev_set_clk_freq ( amdsmi_device_handle device_handle, amdsmi_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 handle `device_handle`, 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 `amdsmi_frequencies_t` returned by `amdsmi_dev_get_gpu_clk_freq()`) corresponding to that bit index will be allowed.

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

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

#### Note

This function requires root access

#### Parameters

in	<i>device_handle</i>	a device handle
in	<i>clk_type</i>	the type of clock for which the set of frequencies will be modified
in	<i>freq_bitmask</i>	A bitmask indicating the indices of the frequencies that are to be enabled (1) and disabled (0). Only the lowest <code>amdsmi_frequencies_t.num_supported</code> bits of this mask are relevant.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

## 6.13 Version Queries

### Functions

- [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_version](#) ([amdsmi\\_version\\_t](#) \*version)  
*Get the build version information for the currently running build of AMDSMI.*
- [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_version\\_str](#) ([amdsmi\\_sw\\_component\\_t](#) component, char \*ver\_str, uint32\_t len)  
*Get the driver version string for the current system.*

### 6.13.1 Detailed Description

These functions provide version information about various subsystems.

### 6.13.2 Function Documentation

#### 6.13.2.1 [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_version](#) ( [amdsmi\\_version\\_t](#) \* version )

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

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

#### Parameters

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

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.13.2.2 [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_version\\_str](#) ( [amdsmi\\_sw\\_component\\_t](#) component, char \* ver\_str, uint32\_t len )

Get the driver version string for the current system.

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

#### Parameters

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

**Note**

`AMDSMI_STATUS_INSUFFICIENT_SIZE` is returned if `len` bytes is not large enough to hold the entire name. In this case, only `len` bytes will be written.

**Returns**

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

## 6.14 Error Queries

### Functions

- `amdsmi_status_t amdsmi_dev_get_ecc_count (amdsmi_device_handle device_handle, amdsmi_gpu_block_t block, amdsmi_error_count_t *ec)`  
Retrieve the error counts for a GPU block.
- `amdsmi_status_t amdsmi_dev_get_ecc_enabled (amdsmi_device_handle device_handle, uint64_t *enabled_blocks)`  
Retrieve the enabled ECC bit-mask.
- `amdsmi_status_t amdsmi_dev_get_ecc_status (amdsmi_device_handle device_handle, amdsmi_gpu_block_t block, amdsmi_ras_err_state_t *state)`  
Retrieve the ECC status for a GPU block.
- `amdsmi_status_t amdsmi_status_string (amdsmi_status_t status, const char **status_string)`  
Get a description of a provided AMDSMI error status.

### 6.14.1 Detailed Description

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

### 6.14.2 Function Documentation

#### 6.14.2.1 `amdsmi_status_t amdsmi_dev_get_ecc_count ( amdsmi_device_handle device_handle, amdsmi_gpu_block_t block, amdsmi_error_count_t * ec )`

Retrieve the error counts for a GPU block.

Given a device handle `device_handle`, an `amdsmi_gpu_block_t block` and a pointer to an `amdsmi_error_count_t ec`, this function will write the error count values for the GPU block indicated by `block` to memory pointed to by `ec`.

#### Parameters

in	<code>device_handle</code>	a device handle
in	<code>block</code>	The block for which error counts should be retrieved
in, out	<code>ec</code>	A pointer to an <code>amdsmi_error_count_t</code> to which the error counts should be written. If this parameter is nullptr, this function will return <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.14.2.2 `amdsmi_status_t amdsmi_dev_get_ecc_enabled ( amdsmi_device_handle device_handle, uint64_t * enabled_blocks )`

Retrieve the enabled ECC bit-mask.



Given a device handle `device_handle`, 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 `amdsmi_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>device_handle</i>	a device handle
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>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided, arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

**6.14.2.3** `amdsmi_status_t amdsmi_dev_get_ecc_status ( amdsmi_device_handle device_handle, amdsmi_gpu_block_t block, amdsmi_ras_err_state_t * state )`

Retrieve the ECC status for a GPU block.

Given a device handle `device_handle`, an `amdsmi_gpu_block_t block` and a pointer to an `amdsmi_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>device_handle</i>	a device handle
in	<i>block</i>	The block for which error counts should be retrieved
in, out	<i>state</i>	A pointer to an <code>amdsmi_ras_err_state_t</code> to which the ECC state should be written. If this parameter is nullptr, this function will return <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided, arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

**6.14.2.4** `amdsmi_status_t amdsmi_status_string ( amdsmi_status_t status, const char ** status_string )`

Get a description of a provided AMDSMI 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

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.15 Performance Counter Functions

### Functions

- `amdsmi_status_t amdsmi_dev_counter_group_supported (amdsmi_device_handle device_handle, amdsmi_event_group_t group)`  
Tell if an event group is supported by a given device.
- `amdsmi_status_t amdsmi_dev_create_counter (amdsmi_device_handle device_handle, amdsmi_event_type_t type, amdsmi_event_handle_t *evnt_handle)`  
Create a performance counter object.
- `amdsmi_status_t amdsmi_dev_destroy_counter (amdsmi_event_handle_t evnt_handle)`  
Deallocate a performance counter object.
- `amdsmi_status_t amdsmi_control_counter (amdsmi_event_handle_t evt_handle, amdsmi_counter_command_t cmd, void *cmd_args)`  
Issue performance counter control commands.
- `amdsmi_status_t amdsmi_read_counter (amdsmi_event_handle_t evt_handle, amdsmi_counter_value_t *value)`  
Read the current value of a performance counter.
- `amdsmi_status_t amdsmi_counter_get_available_counters (amdsmi_device_handle device_handle, amdsmi_event_group_t grp, uint32_t *available)`  
Get the number of currently available counters.

### 6.15.1 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. `amdsmi_dev_counter_group_supported()` can be used to see which event types (`amdsmi_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 :: `amdsmi_counter_get_available_counters()`. 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.

`amdsmi_dev_create_counter()` 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 `amdsmi_control_counter()` by passing `amdsmi_counter_command_t` commands. `AMDSMI_CNTR_CMD_START` starts an event counter and `AMDSMI_CNTR_CMD_STOP` stops a counter. `amdsmi_read_counter()` reads an event counter.

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

## 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 `AMDSMI_CNTR_CMD_START` or calling `amdsmi_read_counter()`, causes AMDSMI (in kernel) to internally record the current absolute counter value
- `amdsmi_read_counter()` 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 `AMDSMI_CNTR_CMD_START` or calling `amdsmi_read_counter()`

Example of event counting sequence:

```

amdsmi_counter_value_t value;

// Determine if AMDSMI_EVTNT_GRP_XGMI is supported for device dv_ind
ret = amdsmi_dev_counter_group_supported(dv_ind,
    AMDSMI_EVTNT_GRP_XGMI);

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

ret = amdsmi_counter_get_available_counters(dv_ind,
    AMDSMI_EVTNT_GRP_XGMI, &counters_available);

// Assuming AMDSMI_EVTNT_GRP_XGMI is supported and there is at least 1
// counter available for AMDSMI_EVTNT_GRP_XGMI on device dv_ind, create
// an event object for an event of group AMDSMI_EVTNT_GRP_XGMI (e.g.,
// AMDSMI_EVTNT_XGMI_0_BEATS_TX) and get the handle
// (amdsmi_event_handle_t).

ret = amdsmi_dev_create_counter(dv_ind,
    AMDSMI_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 = amdsmi_control_counter(evnt_handle,
    AMDSMI_CNTR_CMD_START, NULL);

// Wait...

// Get the number of events since AMDSMI_CNTR_CMD_START was issued:
ret = amdsmi_read_counter(amdsmi_event_handle_t evt_handle, &value)

// Wait...

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

// Stop counting.
ret = amdsmi_control_counter(evnt_handle, AMDSMI_CNTR_CMD_STOP, NULL);

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

```

## 6.15.2 Function Documentation

### 6.15.2.1 `amdsmi_status_t amdsmi_dev_counter_group_supported ( amdsmi_device_handle device_handle, amdsmi_event_group_t group )`

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

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

#### Parameters

in	<code>device_handle</code>	device handle of device being queried
in	<code>group</code>	<code>amdsmi_event_group_t</code> identifier of group for which support is being queried

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

### 6.15.2.2 `amdsmi_status_t amdsmi_dev_create_counter ( amdsmi_device_handle device_handle, amdsmi_event_type_t type, amdsmi_event_handle_t * evnt_handle )`

Create a performance counter object.

Create a performance counter object of type `type` for the device with a device handle of `device_handle`, 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 [amdsmi\\_dev\\_destroy\\_counter\(\)](#) when no longer needed.

#### Note

This function requires root access

#### Parameters

in	<i>device_handle</i>	a device handle
in	<i>type</i>	the <a href="#">amdsmi_event_type_t</a> of performance event to create
in, out	<i>evnt_handle</i>	A pointer to a <a href="#">amdsmi_event_handle_t</a> which will be associated with a newly allocated counter. If this parameter is nullptr, this function will return <a href="#">AMDSMI_STATUS_INVALID</a> if the function is supported with the provided arguments and <a href="#">AMDSMI_STATUS_NOT_SUPPORTED</a> if it is not supported with the provided arguments.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.15.2.3 [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_destroy\\_counter](#) ( [amdsmi\\_event\\_handle\\_t](#) *evnt\_handle* )

Deallocate a performance counter object.

Deallocate the performance counter object with the provided [amdsmi\\_event\\_handle\\_t](#) `evnt_handle`

#### Note

This function requires root access

#### Parameters

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

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.15.2.4 [amdsmi\\_status\\_t](#) [amdsmi\\_control\\_counter](#) ( [amdsmi\\_event\\_handle\\_t](#) *evt\_handle*, [amdsmi\\_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`.

#### Note

This function requires root access

## 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.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.15.2.5 [amdsmi\\_status\\_t](#) [amdsmi\\_read\\_counter](#) ( [amdsmi\\_event\\_handle\\_t](#) *evt\_handle*, [amdsmi\\_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*.

## Note

This function requires root access

## Parameters

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

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.15.2.6 [amdsmi\\_status\\_t](#) [amdsmi\\_counter\\_get\\_available\\_counters](#) ( [amdsmi\\_device\\_handle](#) *device\_handle*, [amdsmi\\_event\\_group\\_t](#) *grp*, [uint32\\_t](#) \* *available* )

Get the number of currently available counters.

Given a device handle *device\_handle*, 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 handle *device\_handle* to the memory that *available* points to.

## Parameters

in	<i>device_handle</i>	a device handle
in	<i>grp</i>	an event device group
in, out	<i>available</i>	A pointer to a <a href="#">uint32_t</a> to which the number of available counters will be written

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail



## 6.16 System Information Functions

### Functions

- `amdsmi_status_t amdsmi_get_compute_process_info (amdsmi_process_info_t *procs, uint32_t *num_items)`  
*Get process information about processes currently using GPU.*
- `amdsmi_status_t amdsmi_get_compute_process_info_by_pid (uint32_t pid, amdsmi_process_info_t *proc)`  
*Get process information about a specific process.*
- `amdsmi_status_t amdsmi_get_compute_process_gpus (uint32_t pid, uint32_t *dv_indices, uint32_t *num_devices)`  
*Get the device indices currently being used by a process.*

### 6.16.1 Detailed Description

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

### 6.16.2 Function Documentation

#### 6.16.2.1 `amdsmi_status_t amdsmi_get_compute_process_info ( amdsmi_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 `amdsmi_process_info_t`'s, of length `*num_items`, this function will write up to `*num_items` instances of `amdsmi_process_info_t` to the memory pointed to by `procs`. These instances contain information about each process utilizing a GPU. If `procs` is not NULL, `num_items` will be updated with the number of processes actually written. If `procs` is NULL, `num_items` will be updated with the number of processes for which there is current process information. Calling this function with `procs` being NULL is a way to determine how much memory should be allocated for when `procs` is not NULL.

#### Parameters

in, out	<code>procs</code>	a pointer to memory provided by the caller to which process information will be written. This may be NULL in which case only <code>num_items</code> will be updated with the number of processes found.
in, out	<code>num_items</code>	A pointer to a <code>uint32_t</code> , which on input, should contain the amount of memory in <code>amdsmi_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>amdsmi_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.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.16.2.2 `amdsmi_status_t amdsmi_get_compute_process_info_by_pid ( uint32_t pid, amdsmi_process_info_t * proc )`

Get process information about a specific process.

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

#### Parameters

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

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.16.2.3 `amdsmi_status_t amdsmi_get_compute_process_gpus ( 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 `device_handles` of length `*num_devices`, this function will write up to `num_devices` device indices to the memory pointed to by `device_handles`. If `device_handles` is not NULL, `num_devices` will be updated with the number of gpu's currently being used by process `pid`. If `device_handles` is NULL, `device_handles` 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 `device_handles` 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>device_handles</code> argument. On output, if <code>device_handles</code> is non-NULL, this will be updated with the number <code>uint32_t</code> 's actually written. If <code>device_handles</code> is NULL, this argument will be updated with the number devices being used.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

## 6.17 XGMI Functions

### Functions

- `amdsmi_status_t amdsmi_dev_xgmi_error_status (amdsmi_device_handle device_handle, amdsmi_xgmi_status_t *status)`  
*Retrieve the XGMI error status for a device.*
- `amdsmi_status_t amdsmi_dev_reset_xgmi_error (amdsmi_device_handle device_handle)`  
*Reset the XGMI error status for a device.*

### 6.17.1 Detailed Description

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

### 6.17.2 Function Documentation

#### 6.17.2.1 `amdsmi_status_t amdsmi_dev_xgmi_error_status ( amdsmi_device_handle device_handle, amdsmi_xgmi_status_t * status )`

Retrieve the XGMI error status for a device.

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

#### Parameters

in	<code>device_handle</code>	a device handle
in, out	<code>status</code>	A pointer to an <code>amdsmi_xgmi_status_t</code> to which the XGMI error state should be written. If this parameter is nullptr, this function will return <code>AMDSMI_STATUS_INVALID</code> if the function is supported with the provided arguments and <code>AMDSMI_STATUS_NOT_SUPPORTED</code> if it is not supported with the provided arguments.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.17.2.2 `amdsmi_status_t amdsmi_dev_reset_xgmi_error ( amdsmi_device_handle device_handle )`

Reset the XGMI error status for a device.

Given a device handle `device_handle`, this function will reset the current XGMI error state `amdsmi_xgmi_status_t` for the device `device_handle` to `amdsmi_xgmi_status_t::AMDSMI_XGMI_STATUS_NO_ERRORS`

**Parameters**

in	<i>device_handle</i>	a device handle
----	----------------------	-----------------

**Returns**

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.18 Hardware Topology Functions

### Functions

- `amdsmi_status_t amdsmi_topo_get_numa_node_number (amdsmi_device_handle device_handle, uint32_t *numa_node)`  
Retrieve the NUMA CPU node number for a device.
- `amdsmi_status_t amdsmi_topo_get_link_weight (amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, uint64_t *weight)`  
Retrieve the weight for a connection between 2 GPUs.
- `amdsmi_status_t amdsmi_get_minmax_bandwidth (amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, uint64_t *min_bandwidth, uint64_t *max_bandwidth)`  
Retrieve minimal and maximal io link bandwidth between 2 GPUs.
- `amdsmi_status_t amdsmi_topo_get_link_type (amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, uint64_t *hops, AMDSMI_IO_LINK_TYPE *type)`  
Retrieve the hops and the connection type between 2 GPUs.
- `amdsmi_status_t amdsmi_is_P2P_accessible (amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, bool *accessible)`  
Return P2P availability status between 2 GPUs.

### 6.18.1 Detailed Description

These functions are used to query Hardware topology.

### 6.18.2 Function Documentation

#### 6.18.2.1 `amdsmi_status_t amdsmi_topo_get_numa_node_number ( amdsmi_device_handle device_handle, uint32_t *numa_node )`

Retrieve the NUMA CPU node number for a device.

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

#### Parameters

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

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.18.2.2 `amdsmi_status_t amdsmi_topo_get_link_weight ( amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, uint64_t * weight )`

Retrieve the weight for a connection between 2 GPUs.

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

#### Parameters

in	<code>device_handle_src</code>	the source device handle
in	<code>device_handle_dst</code>	the destination device handle
in, out	<code>weight</code>	A pointer to an <code>uint64_t</code> to which the weight for the connection should be written.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.18.2.3 `amdsmi_status_t amdsmi_get_minmax_bandwidth ( amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, uint64_t * min_bandwidth, uint64_t * max_bandwidth )`

Retrieve minimal and maximal io link bandwidth between 2 GPUs.

Given a source device handle `device_handle_src` and a destination device handle `device_handle_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	<code>device_handle_src</code>	the source device handle
in	<code>device_handle_dst</code>	the destination device handle
in, out	<code>min_bandwidth</code>	A pointer to an <code>uint64_t</code> to which the minimal bandwidth for the connection should be written.
in, out	<code>max_bandwidth</code>	A pointer to an <code>uint64_t</code> to which the maximal bandwidth for the connection should be written.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.18.2.4 `amdsmi_status_t amdsmi_topo_get_link_type ( amdsmi_device_handle device_handle_src, amdsmi_device_handle device_handle_dst, uint64_t * hops, AMDSMI_IO_LINK_TYPE * type )`

Retrieve the hops and the connection type between 2 GPUs.

Given a source device handle `device_handle_src` and a destination device handle `device_handle_dst`, and a pointer to an `uint64_t` `hops` and a pointer to an `AMDSMI_IO_LINK_TYPE` `type`, this function will write the number of hops and the connection type between the device `device_handle_src` and `device_handle_dst` to the memory pointed to by `hops` and `type`.

## Parameters

in	<i>device_handle_src</i>	the source device handle
in	<i>device_handle_dst</i>	the destination device handle
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 <code>AMD_SMI_IO_LINK_TYPE</code> to which the type for the connection should be written.

## Returns

`amdsmi_status_t` | `AMD_SMI_STATUS_SUCCESS` on success, non-zero on fail

**6.18.2.5** `amdsmi_status_t amsmi_is_P2P_accessible ( amsmi_device_handle device_handle_src, amsmi_device_handle device_handle_dst, bool * accessible )`

Return P2P availability status between 2 GPUs.

Given a source device handle `device_handle_src` and a destination device handle `device_handle_dst`, and a pointer to a `bool` `accessible`, this function will write the P2P connection status between the device `device_handle_src` and `device_handle_dst` to the memory pointed to by `accessible`.

## Parameters

in	<i>device_handle_src</i>	the source device handle
in	<i>device_handle_dst</i>	the destination device handle
in, out	<i>accessible</i>	A pointer to a <code>bool</code> to which the status for the P2P connection availability should be written.

## Returns

`amdsmi_status_t` | `AMD_SMI_STATUS_SUCCESS` on success, non-zero on fail

## 6.19 Supported Functions

### Functions

- `amdsmi_status_t amdsmi_dev_open_supported_func_iterator (amdsmi_device_handle device_handle, amdsmi_func_id_iter_handle_t *handle)`

*Get a function name iterator of supported AMDSMI functions for a device.*

- `amdsmi_status_t amdsmi_dev_open_supported_variant_iterator (amdsmi_func_id_iter_handle_t obj_h, amdsmi_func_id_iter_handle_t *var_iter)`

*Get a variant iterator for a given handle.*

- `amdsmi_status_t amdsmi_next_func_iter (amdsmi_func_id_iter_handle_t handle)`

*Advance a function identifier iterator.*

- `amdsmi_status_t amdsmi_dev_close_supported_func_iterator (amdsmi_func_id_iter_handle_t *handle)`

*Close a variant iterator handle.*

- `amdsmi_status_t amdsmi_get_func_iter_value (amdsmi_func_id_iter_handle_t handle, amdsmi_func_id_t value_t *value)`

*Get the value associated with a function/variant iterator.*

### 6.19.1 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, `::amdsmi_dev_get_temp_metric` may support some types of temperature metrics (e.g., `AMDSMI_TEMP_CRITICAL_HYST`), but not others (e.g., `AMDSMI_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 `::amdsmi_dev_get_temp_metric` example, if variant `AMDSMI_TEMP_CRITICAL_HYST` is supported, perhaps only the sub-variant sensors `::AMDSMI_TEMP_TYPE_EDGE` and `::AMDSMI_TEMP_TYPE_EDGE` are supported, but not `::AMDSMI_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", `AMDSMI_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 AMDSMI types.



```

amdsmi_func_id_iter_handle_t iter_handle, var_iter, sub_var_iter;
amdsmi_func_id_value_t value;
amdsmi_status_t err;
amdsmi_device_handle device;

// Get the device handle via amdsmi_get_device_handles()
// ... ...

std::cout << "Supported AMDSMI Functions:" << std::endl; *
err = amdsmi_dev_open_supported_func_iterator(device, &iter_handle)
    ;

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

    err = amdsmi_dev_open_supported_variant_iterator(iter_handle,
        &var_iter);
    if (err != AMDSMI_STATUS_NO_DATA) {
        std::cout << "\tVariants/Monitors: ";
        while (1) {
            err = amdsmi_get_func_iter_value(var_iter, &value);
            if (value.id == AMDSMI_DEFAULT_VARIANT) {
                std::cout << "Default Variant ";
            } else {
                std::cout << value.id;
            }
            std::cout << " (";

            err =
                amdsmi_dev_open_supported_variant_iterator(var_iter, &
sub_var_iter);
            if (err != AMDSMI_STATUS_NO_DATA) {

                while (1) {
                    err = amdsmi_get_func_iter_value(sub_var_iter, &value);
                    std::cout << value.id << ", ";

                    err = amdsmi_next_func_iter(sub_var_iter);

                    if (err == AMDSMI_STATUS_NO_DATA) {
                        break;
                    }
                }
                err = amdsmi_dev_close_supported_func_iterator(&
sub_var_iter);
            }

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

            err = amdsmi_next_func_iter(var_iter);

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

        err = amdsmi_dev_close_supported_func_iterator(&var_iter);
    }

    err = amdsmi_next_func_iter(iter_handle);

    if (err == AMDSMI_STATUS_NO_DATA) {
        break;
    }
}
err = amdsmi_dev_close_supported_func_iterator(&iter_handle);
}

```

## 6.19.2 Function Documentation

### 6.19.2.1 amdsmi\_status\_t amdsmi\_dev\_open\_supported\_func\_iterator ( amdsmi\_device\_handle device\_handle, amdsmi\_func\_id\_iter\_handle\_t \* handle )

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

Given a device handle `device_handle`, 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 `device_handle` as an argument, [amdsmi\\_dev\\_open\\_supported\\_func\\_iterator](#) itself will not be among the functions listed as supported. This is because [amdsmi\\_dev\\_open\\_supported\\_func\\_iterator](#) does not depend on hardware or driver support and should always be supported.

#### Parameters

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

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.19.2.2** `amdsmi_status_t amdsmi_dev_open_supported_variant_iterator ( amdsmi_func_id_iter_handle_t obj_h, amdsmi_func_id_iter_handle_t * var_iter )`

Get a variant iterator for a given handle.

Given a [amdsmi\\_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 [amdsmi\\_dev\\_open\\_supported\\_func\\_iterator](#), or it may be a variant itself (from a call to [amdsmi\\_dev\\_open\\_supported\\_variant\\_iterator](#)), 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 [amdsmi\\_dev\\_close\\_supported\\_func\\_iterator](#) should be called on the returned iterator handle `var_iter` when it is no longer needed.

#### Parameters

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

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

**6.19.2.3** `amdsmi_status_t amdsmi_next_func_iter ( amdsmi_func_id_iter_handle_t handle )`

Advance a function identifier iterator.

Given a function id iterator handle ([amdsmi\\_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, [AMDSMI\\_STATUS\\_NO\\_DATA](#) is returned.

#### Parameters

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

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.19.2.4 [amdsmi\\_status\\_t](#) [amdsmi\\_dev\\_close\\_supported\\_func\\_iterator](#) ( [amdsmi\\_func\\_id\\_iter\\_handle\\_t](#) \* *handle* )

Close a variant iterator handle.

Given a pointer to an [amdsmi\\_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
----	---------------	--

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.19.2.5 [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_func\\_iter\\_value](#) ( [amdsmi\\_func\\_id\\_iter\\_handle\\_t](#) *handle*, [amdsmi\\_func\\_id\\_value\\_t](#) \* *value* )

Get the value associated with a function/variant iterator.

Given an [amdsmi\\_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="#">amdsmi_func_id_value_t</a> provided by the caller to which this function will write the value associated with <i>handle</i>

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.20 Event Notification Functions

### Functions

- `amdsmi_status_t amdsmi_init_event_notification (amdsmi_device_handle device_handle)`  
*Prepare to collect event notifications for a GPU.*
- `amdsmi_status_t amdsmi_set_event_notification_mask (amdsmi_device_handle device_handle, uint64_t mask)`  
*Specify which events to collect for a device.*
- `amdsmi_status_t amdsmi_get_event_notification (int timeout_ms, uint32_t *num_elem, amdsmi_evt_notification_data_t *data)`  
*Collect event notifications, waiting a specified amount of time.*
- `amdsmi_status_t amdsmi_stop_event_notification (amdsmi_device_handle device_handle)`  
*Close any file handles and free any resources used by event notification for a GPU.*

### 6.20.1 Detailed Description

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

### 6.20.2 Function Documentation

#### 6.20.2.1 `amdsmi_status_t amdsmi_init_event_notification ( amdsmi_device_handle device_handle )`

Prepare to collect event notifications for a GPU.

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

#### Parameters

<code>device_handle</code>	a device handle corresponding to the device on which to listen for events
----------------------------	---

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.20.2.2 `amdsmi_status_t amdsmi_set_event_notification_mask ( amdsmi_device_handle device_handle, uint64_t mask )`

Specify which events to collect for a device.

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

## Parameters

<i>device_handle</i>	a device handle corresponding to the device on which to listen for events
<i>mask</i>	Bitmask generated by OR'ing 1 or more elements of <a href="#">amdsmi_evt_notification_type_t</a> indicating which event types to listen for, where the <a href="#">amdsmi_evt_notification_type_t</a> 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 AMDSMI_EVT_NOTIF_VMFAULT (which has a value of 1) and AMDSMI_EVT_NOTIF_THERMAL_THROTTLE event (which has a value of 2).

## Note

[AMDSMI\\_STATUS\\_INIT\\_ERROR](#) is returned if [amdsmi\\_init\\_event\\_notification\(\)](#) has not been called before a call to this function

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

### 6.20.2.3 [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_event\\_notification](#) ( *int* *timeout\_ms*, *uint32\_t* \* *num\_elem*, [amdsmi\\_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 [amdsmi\\_evt\\_notification\\_data\\_t](#)'s *data* with a length (in [amdsmi\\_evt\\_notification\\_data\\_t](#)'s, also specified by the caller) in the memory location pointed to by *num\_elem*, this function will collect [amdsmi\\_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 [amdsmi\\_init\\_event\\_notification\(\)](#) and :: [amdsmi\\_set\\_event\\_notification\\_mask\(\)](#). This function polls for the occurrence of the events on the respective devices that were previously specified by :: [amdsmi\\_set\\_event\\_notification\\_mask\(\)](#).

## 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 <i>uint32_t</i> , provided by the caller. On input, this value tells how many <a href="#">amdsmi_evt_notification_data_t</a> elements are being provided by the caller with <i>data</i> . On output, the location pointed to by <i>num_elem</i> 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 <i>num_elem</i> <a href="#">amdsmi_evt_notification_data_t</a> to which this function may safely write. If there are events found, up to <i>num_elem</i> event items will be written to <i>data</i> .

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.20.2.4 `amdsmi_status_t` `amdsmi_stop_event_notification` ( `amdsmi_device_handle` *device\_handle* )

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 handle `device_handle` will be free with this function. This includes freeing any memory and closing file handles. This should be called for every call to [amdsmi\\_↵  
\\_init\\_event\\_notification\(\)](#)

##### Parameters

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

##### Returns

`amdsmi_status_t` | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.21 SW Version Information

### Functions

- [amdsmi\\_status\\_t amdsmi\\_get\\_driver\\_version](#) ([amdsmi\\_device\\_handle](#) device\_handle, int \*length, char \*version)

*Returns the driver version information.*

### 6.21.1 Detailed Description

### 6.21.2 Function Documentation

#### 6.21.2.1 [amdsmi\\_status\\_t amdsmi\\_get\\_driver\\_version](#) ( [amdsmi\\_device\\_handle](#) device\_handle, int \* length, char \* version )

Returns the driver version information.

#### Parameters

in	<i>device_handle</i>	Device which to query
in, out	<i>length</i>	As input parameter length of the user allocated string buffer. As output parameter length of the returned string buffer.
out	<i>version</i>	Version information in string format. Must be allocated by user.

#### Returns

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.22 ASIC & Board Static Information

### Functions

- `amdsmi_status_t amdsmi_get_asic_info (amdsmi_device_handle device_handle, amdsmi_asic_info_t *info)`  
*Returns the ASIC information for the device.*
- `amdsmi_status_t amdsmi_get_board_info (amdsmi_device_handle device_handle, amdsmi_board_info_t *info)`  
*Returns the board part number and board information for the requested device.*
- `amdsmi_status_t amdsmi_get_power_cap_info (amdsmi_device_handle device_handle, uint32_t sensor_ind, amdsmi_power_cap_info_t *info)`  
*Returns the power caps as currently configured in the system.*
- `amdsmi_status_t amdsmi_get_xgmi_info (amdsmi_device_handle device_handle, amdsmi_xgmi_info_t *info)`  
*Returns XGMI information for the GPU.*
- `amdsmi_status_t amdsmi_get_caps_info (amdsmi_device_handle device_handle, amdsmi_gpu_caps_t *info)`  
*Returns the device capabilities as currently configured in the system.*

### 6.22.1 Detailed Description

### 6.22.2 Function Documentation

#### 6.22.2.1 `amdsmi_status_t amdsmi_get_asic_info ( amdsmi_device_handle device_handle, amdsmi_asic_info_t * info )`

Returns the ASIC information for the device.

This function returns ASIC information such as the product name, the family, the vendor ID, the subvendor ID, the device ID, the revision ID and the serial number.

#### Parameters

in	<code>device_handle</code>	Device which to query
out	<code>info</code>	Reference to static asic information structure. Must be allocated by user.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.22.2.2 `amdsmi_status_t amdsmi_get_board_info ( amdsmi_device_handle device_handle, amdsmi_board_info_t * info )`

Returns the board part number and board information for the requested device.



## Parameters

in	<i>device_handle</i>	Device which to query
out	<i>info</i>	Reference to board info structure. Must be allocated by user.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.22.2.3 `amdsmi_status_t amdsmi_get_power_cap_info ( amdsmi_device_handle device_handle, uint32_t sensor_ind, amdsmi_power_cap_info_t * info )`

Returns the power caps as currently configured in the system.

## Parameters

in	<i>device_handle</i>	Device which to query
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.
out	<i>info</i>	Reference to power caps information structure. Must be allocated by user.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.22.2.4 `amdsmi_status_t amdsmi_get_xgmi_info ( amdsmi_device_handle device_handle, amdsmi_xgmi_info_t * info )`

Returns XGMI information for the GPU.

## Parameters

in	<i>device_handle</i>	Device which to query
out	<i>info</i>	Reference to xgmi information structure. Must be allocated by user.

## Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.22.2.5 `amdsmi_status_t amdsmi_get_caps_info ( amdsmi_device_handle device_handle, amdsmi_gpu_caps_t * info )`

Returns the device capabilities as currently configured in the system.

**Parameters**

in	<i>device_handle</i>	Device which to query
out	<i>info</i>	Reference to caps information structure. Must be allocated by user.

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.23 Firmware & VBIOS queries

### Functions

- `amdsmi_status_t amdsmi_get_fw_info (amdsmi_device_handle device_handle, amdsmi_fw_info_t *info)`  
*Returns the firmware versions running on the device.*
- `amdsmi_status_t amdsmi_get_vbios_info (amdsmi_device_handle device_handle, amdsmi_vbios_info_t *info)`  
*Returns the static information for the vBIOS on the device.*

#### 6.23.1 Detailed Description

#### 6.23.2 Function Documentation

##### 6.23.2.1 `amdsmi_status_t amdsmi_get_fw_info ( amdsmi_device_handle device_handle, amdsmi_fw_info_t * info )`

Returns the firmware versions running on the device.

##### Parameters

in	<i>device_handle</i>	Device which to query
out	<i>info</i>	Reference to the fw info. Must be allocated by user.

##### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

##### 6.23.2.2 `amdsmi_status_t amdsmi_get_vbios_info ( amdsmi_device_handle device_handle, amdsmi_vbios_info_t * info )`

Returns the static information for the vBIOS on the device.

##### Parameters

in	<i>device_handle</i>	Device which to query
out	<i>info</i>	Reference to static vBIOS information. Must be allocated by user.

##### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

## 6.24 GPU Monitoring

### Functions

- `amdsmi_status_t amdsmi_get_gpu_activity (amdsmi_device_handle device_handle, amdsmi_engine_usage_t *info)`  
Returns the current usage of the GPU engines (GFX, MM and MEM). Each usage is reported as a percentage from 0-100%.
- `amdsmi_status_t amdsmi_get_power_measure (amdsmi_device_handle device_handle, amdsmi_power_measure_t *info)`  
Returns the current power and voltage of the GPU. The voltage is in units of mV and the power in units of W.
- `amdsmi_status_t amdsmi_get_clock_measure (amdsmi_device_handle device_handle, amdsmi_clk_type_t clk_type, amdsmi_clk_measure_t *info)`  
Returns the measurements of the clocks in the GPU for the GFX and multimedia engines and Memory. This call reports the averages over 1s in MHz.
- `amdsmi_status_t amdsmi_get_vram_usage (amdsmi_device_handle device_handle, amdsmi_vram_info_t *info)`  
Returns the VRAM usage (both total and used memory) in MegaBytes.

### 6.24.1 Detailed Description

### 6.24.2 Function Documentation

#### 6.24.2.1 `amdsmi_status_t amdsmi_get_gpu_activity ( amdsmi_device_handle device_handle, amdsmi_engine_usage_t * info )`

Returns the current usage of the GPU engines (GFX, MM and MEM). Each usage is reported as a percentage from 0-100%.

#### Parameters

in	<code>device_handle</code>	Device which to query
out	<code>info</code>	Reference to the gpu engine usage structure. Must be allocated by user.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.24.2.2 `amdsmi_status_t amdsmi_get_power_measure ( amdsmi_device_handle device_handle, amdsmi_power_measure_t * info )`

Returns the current power and voltage of the GPU. The voltage is in units of mV and the power in units of W.

#### Parameters

in	<code>device_handle</code>	Device which to query
out	<code>info</code>	Reference to the gpu power structure. Must be allocated by user.

## Returns

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.24.2.3 `amdsmi_status_t amsmi_get_clock_measure ( amsmi_device_handle device_handle, amsmi_clk_type_t clk_type, amsmi_clk_measure_t * info )`

Returns the measurements of the clocks in the GPU for the GFX and multimedia engines and Memory. This call reports the averages over 1s in MHz.

## Parameters

in	<i>device_handle</i>	Device which to query
in	<i>clk_type</i>	Enum representing the clock type to query.
out	<i>info</i>	Reference to the gpu clock structure. Must be allocated by user.

## Returns

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

#### 6.24.2.4 `amdsmi_status_t amsmi_get_vram_usage ( amsmi_device_handle device_handle, amsmi_vram_info_t * info )`

Returns the VRAM usage (both total and used memory) in MegaBytes.

## Parameters

in	<i>device_handle</i>	Device which to query
out	<i>info</i>	Reference to vram information. Must be allocated by user.

## Returns

[amdsmi\\_status\\_t](#) | [AMD\\_SMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.25 Power Management

### Functions

- [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_target\\_frequency\\_range](#) ([amdsmi\\_device\\_handle](#) device\_handle, [amdsmi\\_clk\\_type\\_t](#) clk\_type, [amdsmi\\_frequency\\_range\\_t](#) \*range)

*Returns current and supported frequency range for the specified clock type.*

### 6.25.1 Detailed Description

### 6.25.2 Function Documentation

6.25.2.1 [amdsmi\\_status\\_t](#) [amdsmi\\_get\\_target\\_frequency\\_range](#) ( [amdsmi\\_device\\_handle](#) device\_handle, [amdsmi\\_clk\\_type\\_t](#) clk\_type, [amdsmi\\_frequency\\_range\\_t](#) \* range )

Returns current and supported frequency range for the specified clock type.

#### Parameters

in	<i>device_handle</i>	Device which to query
in	<i>clk_type</i>	Clock type for which to get current and supported frequency range.
out	<i>range</i>	Reference to frequency range structure. Must be allocated by user.

#### Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 6.26 Process information

### Functions

- `amdsmi_status_t amdsmi_get_process_list (amdsmi_device_handle device_handle, amdsmi_process_handle *list, uint32_t *max_processes)`  
Returns the list of processes running on a given GPU including itself.
- `amdsmi_status_t amdsmi_get_process_info (amdsmi_device_handle device_handle, amdsmi_process_handle process, amdsmi_proc_info_t *info)`  
Returns the process information of a given process. Engine usage show how much time the process spend using these engines in ns.

### 6.26.1 Detailed Description

### 6.26.2 Function Documentation

#### 6.26.2.1 `amdsmi_status_t amdsmi_get_process_list ( amdsmi_device_handle device_handle, amdsmi_process_handle * list, uint32_t * max_processes )`

Returns the list of processes running on a given GPU including itself.

#### Note

The user provides a buffer to store the list and the maximum number of processes that can be returned. If the user sets max\_processes to 0, the total number of processes will be returned.

#### Parameters

in	<i>device_handle</i>	Device which to query
out	<i>list</i>	Reference to a user-provided buffer where the process list will be returned. This buffer must contain at least max_processes entries of type smi_process_handle. Must be allocated by user.
in, out	<i>max_processes</i>	Reference to the size of the list buffer in number of elements. Returns the return number of elements in list or the number of running processes if equal to 0.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail

#### 6.26.2.2 `amdsmi_status_t amdsmi_get_process_info ( amdsmi_device_handle device_handle, amdsmi_process_handle process, amdsmi_proc_info_t * info )`

Returns the process information of a given process. Engine usage show how much time the process spend using these engines in ns.

**Parameters**

in	<i>device_handle</i>	Device which to query
in	<i>process</i>	Handle of process to query.
out	<i>info</i>	Reference to a process information structure where to return information. Must be allocated by user.

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail



## 6.27 ECC information

### Functions

- `amdsmi_status_t amdsmi_get_ecc_error_count (amdsmi_device_handle device_handle, amdsmi_error_count_t *ec)`

*Returns the number of ECC errors (correctable and uncorrectable) in the given GPU.*

### 6.27.1 Detailed Description

### 6.27.2 Function Documentation

6.27.2.1 `amdsmi_status_t amdsmi_get_ecc_error_count ( amdsmi_device_handle device_handle, amdsmi_error_count_t * ec )`

Returns the number of ECC errors (correctable and uncorrectable) in the given GPU.

#### Parameters

in	<code>device_handle</code>	Device which to query
out	<code>ec</code>	Reference to ecc error count structure. Must be allocated by user.

#### Returns

`amdsmi_status_t` | `AMDSMI_STATUS_SUCCESS` on success, non-zero on fail



## Chapter 7

# Data Structure Documentation

### 7.1 amd\_metrics\_table\_header\_t Struct Reference

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

```
#include <amdsmi.h>
```

#### 7.1.1 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:

- [amdsmi.h](#)

### 7.2 amdsmi\_asic\_info\_t Struct Reference

#### Data Fields

- char **market\_name** [AMDSMI\_MAX\_STRING\_LENGTH]
- uint32\_t [family](#)
- uint32\_t **vendor\_id**
- uint32\_t **subvendor\_id**
- uint64\_t **device\_id**
- uint32\_t **rev\_id**
- char **asic\_serial** [AMDSMI\_NORMAL\_STRING\_LENGTH]

### 7.2.1 Field Documentation

#### 7.2.1.1 uint32\_t amdsmi\_asic\_info\_t::family

Has zero value

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

- [amdsmi.h](#)

## 7.3 amdsmi\_bdf\_t Union Reference

### Data Fields

- struct {
  - uint64\_t **function\_number**: 3
  - uint64\_t **device\_number**: 5
  - uint64\_t **bus\_number**: 8
  - uint64\_t **domain\_number**: 48
- };
- uint64\_t **as\_uint**

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

- [amdsmi.h](#)

## 7.4 amdsmi\_board\_info\_t Struct Reference

### Data Fields

- uint64\_t **serial\_number**
- bool **is\_master**
- char **model\_number** [AMDSMI\_NORMAL\_STRING\_LENGTH]
- char **product\_serial** [AMDSMI\_NORMAL\_STRING\_LENGTH]
- char **fru\_id** [AMDSMI\_NORMAL\_STRING\_LENGTH]
- char **product\_name** [AMDSMI\_PRODUCT\_NAME\_LENGTH]
- char **manufacturer\_name** [AMDSMI\_NORMAL\_STRING\_LENGTH]

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

- [amdsmi.h](#)

## 7.5 amdsmi\_clk\_measure\_t Struct Reference

### Data Fields

- uint32\_t **cur\_clk**
- uint32\_t **avg\_clk**
- uint32\_t **min\_clk**
- uint32\_t **max\_clk**
- uint32\_t **reserved** [4]

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

- [amdsmi.h](#)

## 7.6 amdsmi\_counter\_value\_t Struct Reference

```
#include <amdsmi.h>
```

### Data Fields

- uint64\_t [value](#)  
*Counter value.*
- uint64\_t [time\\_enabled](#)
- uint64\_t [time\\_running](#)

### 7.6.1 Detailed Description

Counter value

### 7.6.2 Field Documentation

#### 7.6.2.1 uint64\_t amdsmi\_counter\_value\_t::time\_enabled

Time that the counter was enabled (in nanoseconds)

#### 7.6.2.2 uint64\_t amdsmi\_counter\_value\_t::time\_running

Time that the counter was running (in nanoseconds)

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

- [amdsmi.h](#)

## 7.7 amdsmi\_engine\_usage\_t Struct Reference

### Data Fields

- uint32\_t **gfx\_activity**
- uint32\_t **umc\_activity**
- uint32\_t **mm\_activity** [AMDSMI\_MAX\_MM\_IP\_COUNT]
- uint32\_t **reserved** [6]

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

- [amdsmi.h](#)

## 7.8 amdsmi\_error\_count\_t Struct Reference

This structure holds error counts.

```
#include <amdsmi.h>
```

### Data Fields

- uint64\_t [correctable\\_count](#)  
*Accumulated correctable errors.*
- uint64\_t [uncorrectable\\_count](#)  
*Accumulated uncorrectable errors.*
- uint64\_t **reserved** [2]

### 7.8.1 Detailed Description

This structure holds error counts.

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

- [amdsmi.h](#)

## 7.9 amdsmi\_evt\_notification\_data\_t Struct Reference

```
#include <amdsmi.h>
```

### Data Fields

- [amdsmi\\_device\\_handle](#) device\_handle  
*Handler of device that corresponds to the event.*
- [amdsmi\\_evt\\_notification\\_type\\_t](#) event  
*Event type.*
- char [message](#) [MAX\_EVENT\_NOTIFICATION\_MSG\_SIZE]  
*Event message.*

### 7.9.1 Detailed Description

Event notification data returned from event notification API

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

- [amdsmi.h](#)

## 7.10 amdsmi\_freq\_volt\_region\_t Struct Reference

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

```
#include <amdsmi.h>
```

### Data Fields

- [amdsmi\\_range\\_t freq\\_range](#)  
*The frequency range for this VDDC Curve point.*
- [amdsmi\\_range\\_t volt\\_range](#)  
*The voltage range for this VDDC Curve point.*

### 7.10.1 Detailed Description

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

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

- [amdsmi.h](#)

## 7.11 amdsmi\_frequencies\_t Struct Reference

This structure holds information about clock frequencies.

```
#include <amdsmi.h>
```

### Data Fields

- [uint32\\_t num\\_supported](#)
- [uint32\\_t current](#)
- [uint64\\_t frequency](#) [[AMDSMI\\_MAX\\_NUM\\_FREQUENCIES](#)]

### 7.11.1 Detailed Description

This structure holds information about clock frequencies.

### 7.11.2 Field Documentation

#### 7.11.2.1 `uint32_t amdsmi_frequencies_t::num_supported`

The number of supported frequencies

#### 7.11.2.2 `uint32_t amdsmi_frequencies_t::current`

The current frequency index

#### 7.11.2.3 `uint64_t amdsmi_frequencies_t::frequency[AMDSMI_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:

- [amdsmi.h](#)

## 7.12 `amdsmi_frequency_range_t` Struct Reference

### Data Fields

- [amdsmi\\_range\\_t supported\\_freq\\_range](#)
- [amdsmi\\_range\\_t current\\_freq\\_range](#)
- `uint32_t reserved` [8]

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

- [amdsmi.h](#)

## 7.13 `amdsmi_func_id_value_t` Union Reference

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

```
#include <amdsmi.h>
```



## Data Fields

- uint64\_t **id**  
*uint64\_t representation of value*
- const char \* **name**  
*name string (applicable to functions only)*
- union {
  - [amdsmi\\_memory\\_type\\_t](#) **memory\_type**  
    < *Used for [amdsmi\\_memory\\_type\\_t](#) variants*
  - [amdsmi\\_temperature\\_metric\\_t](#) **temp\_metric**  
    *Used for [amdsmi\\_event\\_type\\_t](#) variants.*
  - [amdsmi\\_event\\_type\\_t](#) **evnt\_type**  
    *Used for [amdsmi\\_event\\_group\\_t](#) variants.*
  - [amdsmi\\_event\\_group\\_t](#) **evnt\_group**  
    *Used for [amdsmi\\_clk\\_type\\_t](#) variants.*
  - [amdsmi\\_clk\\_type\\_t](#) **clk\_type**  
    *Used for [amdsmi\\_fw\\_block\\_t](#) variants.*
  - [amdsmi\\_fw\\_block\\_t](#) **fw\_block**  
    *Used for [amdsmi\\_gpu\\_block\\_t](#) variants.*
  - [amdsmi\\_gpu\\_block\\_t](#) **gpu\_block\_type**

};

### 7.13.1 Detailed Description

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

### 7.13.2 Field Documentation

#### 7.13.2.1 [amdsmi\\_memory\\_type\\_t](#) [amdsmi\\_func\\_id\\_value\\_t::memory\\_type](#)

< Used for [amdsmi\\_memory\\_type\\_t](#) variants

Used for [amdsmi\\_temperature\\_metric\\_t](#) variants

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

- [amdsmi.h](#)

## 7.14 amdsmi\_fw\_info\_t Struct Reference

### Data Fields

- uint8\_t **num\_fw\_info**
- struct {
  - [amdsmi\\_fw\\_block\\_t](#) **fw\_id**
  - uint64\_t **fw\_version**
  - uint64\_t **reserved** [2]
- } **fw\_info\_list** [FW\_ID\_\_MAX]
- uint32\_t **reserved** [7]

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

- [amdsmi.h](#)

## 7.15 amdsmi\_gpu\_caps\_t Struct Reference

```
#include <amdsmi.h>
```

### Data Fields

- struct {
  - uint32\_t **gfxip\_major**
  - uint32\_t **gfxip\_minor**
  - uint16\_t **gfxip\_cu\_count**
  - uint32\_t **reserved** [5]
- **gfx**
- struct {
  - uint8\_t **mm\_ip\_count**
  - uint8\_t **mm\_ip\_list** [AMDSMI\_MAX\_MM\_IP\_COUNT]
  - uint32\_t **reserved** [5]
- **mm**
- bool **ras\_supported**
- uint8\_t **max\_vf\_num**
- uint32\_t **gfx\_ip\_count**
- uint32\_t **dma\_ip\_count**

### 7.15.1 Detailed Description

GPU Capability info

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

- [amdsmi.h](#)

## 7.16 amdsmi\_gpu\_metrics\_t Struct Reference

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

- [amdsmi.h](#)

## 7.17 amdsmi\_od\_vddc\_point\_t Struct Reference

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

```
#include <amdsmi.h>
```

## Data Fields

- [uint64\\_t frequency](#)  
*Frequency coordinate (in Hz)*
- [uint64\\_t voltage](#)  
*Voltage coordinate (in mV)*

### 7.17.1 Detailed Description

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

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

- [amdsmi.h](#)

## 7.18 amdsmi\_od\_volt\_curve\_t Struct Reference

```
#include <amdsmi.h>
```

## Data Fields

- [amdsmi\\_od\\_vddc\\_point\\_t vc\\_points](#) [[AMDSMI\\_NUM\\_VOLTAGE\\_CURVE\\_POINTS](#)]

### 7.18.1 Detailed Description

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

### 7.18.2 Field Documentation

**7.18.2.1** [amdsmi\\_od\\_vddc\\_point\\_t](#) [amdsmi\\_od\\_volt\\_curve\\_t::vc\\_points](#) [[AMDSMI\\_NUM\\_VOLTAGE\\_CURVE\\_POINTS](#)]↔  
INTS]

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

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

- [amdsmi.h](#)

## 7.19 amdsmi\_od\_volt\_freq\_data\_t Struct Reference

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

```
#include <amdsmi.h>
```

## Data Fields

- [amdsmi\\_range\\_t curr\\_sclk\\_range](#)  
*The current SCLK frequency range.*
- [amdsmi\\_range\\_t curr\\_mclk\\_range](#)  
*(upper bound only)*
- [amdsmi\\_range\\_t sclk\\_freq\\_limits](#)  
*The range possible of SCLK values.*
- [amdsmi\\_range\\_t mclk\\_freq\\_limits](#)  
*The range possible of MCLK values.*
- [amdsmi\\_od\\_volt\\_curve\\_t curve](#)  
*The current voltage curve.*
- [uint32\\_t num\\_regions](#)  
*The number of voltage curve regions.*

### 7.19.1 Detailed Description

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

### 7.19.2 Field Documentation

#### 7.19.2.1 [amdsmi\\_range\\_t](#) [amdsmi\\_od\\_volt\\_freq\\_data\\_t::curr\\_mclk\\_range](#)

(upper bound only)

The current MCLK frequency range;

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

- [amdsmi.h](#)

## 7.20 [amdsmi\\_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 <amdsmi.h>
```

## Data Fields

- [amdsmi\\_frequencies\\_t transfer\\_rate](#)
- [uint32\\_t lanes](#) [[AMDSMI\\_MAX\\_NUM\\_FREQUENCIES](#)]

### 7.20.1 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.

## 7.20.2 Field Documentation

### 7.20.2.1 amdsmi\_frequencies\_t amdsmi\_pcie\_bandwidth\_t::transfer\_rate

Transfer rates (T/s) that are possible

### 7.20.2.2 uint32\_t amdsmi\_pcie\_bandwidth\_t::lanes[AMDSMI\_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:

- [amdsmi.h](#)

## 7.21 amdsmi\_pcie\_info\_t Struct Reference

This structure holds pcie info.

```
#include <amdsmi.h>
```

### Data Fields

- uint16\_t **pcie\_lanes**
- uint32\_t **pcie\_speed**
- uint32\_t **reserved** [6]

### 7.21.1 Detailed Description

This structure holds pcie info.

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

- [amdsmi.h](#)

## 7.22 amdsmi\_power\_cap\_info\_t Struct Reference

### Data Fields

- uint64\_t **power\_cap**
- uint64\_t **default\_power\_cap**
- uint64\_t **dpm\_cap**
- uint64\_t **min\_power\_cap**
- uint64\_t **max\_power\_cap**
- uint64\_t **reserved** [3]

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

- [amdsmi.h](#)

## 7.23 amdsmi\_power\_measure\_t Struct Reference

### Data Fields

- uint32\_t **average\_socket\_power**
- uint64\_t **energy\_accumulator**
- uint32\_t **voltage\_gfx**
- uint32\_t **voltage\_soc**
- uint32\_t **voltage\_mem**
- uint32\_t **power\_limit**
- uint32\_t **reserved** [9]

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

- [amdsmi.h](#)

## 7.24 amdsmi\_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 <amdsmi.h>
```

### Data Fields

- [amdsmi\\_bit\\_field\\_t](#) **available\_profiles**
- [amdsmi\\_power\\_profile\\_preset\\_masks\\_t](#) **current**
- uint32\_t **num\_profiles**

### 7.24.1 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.

### 7.24.2 Field Documentation

#### 7.24.2.1 amdsmi\_bit\_field\_t amdsmi\_power\_profile\_status\_t::available\_profiles

Which profiles are supported by this system

#### 7.24.2.2 amdsmi\_power\_profile\_preset\_masks\_t amdsmi\_power\_profile\_status\_t::current

Which power profile is currently active

## 7.24.2.3 uint32\_t amdsmi\_power\_profile\_status\_t::num\_profiles

How many power profiles are available

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

- [amdsmi.h](#)

## 7.25 amdsmi\_proc\_info\_t Struct Reference

## Data Fields

- char **name** [AMDSMI\_NORMAL\_STRING\_LENGTH]
- amdsmi\_process\_handle **pid**
- uint64\_t **mem**
- struct {
  - uint64\_t **gfx**
  - uint64\_t **compute**
  - uint64\_t **dma**
  - uint64\_t **enc**
  - uint64\_t **dec**
 } [engine\\_usage](#)
- struct {
  - uint64\_t **gtt\_mem**
  - uint64\_t **cpu\_mem**
  - uint64\_t **vram\_mem**
 } [memory\\_usage](#)
- char [container\\_name](#) [AMDSMI\_NORMAL\_STRING\_LENGTH]
- uint32\_t **reserved** [10]

## 7.25.1 Field Documentation

## 7.25.1.1 struct { ... } amdsmi\_proc\_info\_t::engine\_usage

in bytes

## 7.25.1.2 struct { ... } amdsmi\_proc\_info\_t::memory\_usage

How much time the process spend using these engines in ns

## 7.25.1.3 char amdsmi\_proc\_info\_t::container\_name[AMDSMI\_NORMAL\_STRING\_LENGTH]

in bytes

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

- [amdsmi.h](#)

## 7.26 amdsmi\_process\_info\_t Struct Reference

This structure contains information specific to a process.

```
#include <amdsmi.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.*

### 7.26.1 Detailed Description

This structure contains information specific to a process.

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

- [amdsmi.h](#)

## 7.27 amdsmi\_range\_t Struct Reference

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

```
#include <amdsmi.h>
```

### Data Fields

- uint64\_t [lower\\_bound](#)  
*Lower bound of range.*
- uint64\_t [upper\\_bound](#)  
*Upper bound of range.*
- uint64\_t **reserved** [2]

### 7.27.1 Detailed Description

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

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

- [amdsmi.h](#)



## 7.28 amdsmi\_retired\_page\_record\_t Struct Reference

Reserved Memory Page Record.

```
#include <amdsmi.h>
```

### Data Fields

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

### 7.28.1 Detailed Description

Reserved Memory Page Record.

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

- [amdsmi.h](#)

## 7.29 amdsmi\_utilization\_counter\_t Struct Reference

The utilization counter data.

```
#include <amdsmi.h>
```

### Data Fields

- [AMDSMI\\_UTILIZATION\\_COUNTER\\_TYPE type](#)  
*Utilization counter type.*
- [uint64\\_t value](#)  
*Utilization counter value.*

### 7.29.1 Detailed Description

The utilization counter data.

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

- [amdsmi.h](#)

## 7.30 amdsmi\_vbios\_info\_t Struct Reference

### Data Fields

- char **name** [AMDSMI\_MAX\_STRING\_LENGTH]
- uint32\_t **vbios\_version**
- char **build\_date** [AMDSMI\_MAX\_DATE\_LENGTH]
- char **part\_number** [AMDSMI\_MAX\_STRING\_LENGTH]
- char **vbios\_version\_string** [AMDSMI\_NORMAL\_STRING\_LENGTH]
- uint32\_t **reserved** [15]

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

- [amdsmi.h](#)

## 7.31 amdsmi\_version\_t Struct Reference

This structure holds version information.

```
#include <amdsmi.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.*
- uint32\_t **reserved** [4]

### 7.31.1 Detailed Description

This structure holds version information.

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

- [amdsmi.h](#)

## 7.32 amdsmi\_vram\_info\_t Struct Reference

### Data Fields

- uint32\_t **vram\_total**
- uint32\_t **vram\_used**

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

- [amdsmi.h](#)

## 7.33 amdsmi\_xgmi\_info\_t Struct Reference

### Data Fields

- uint8\_t **xgmi\_lanes**
- uint64\_t **xgmi\_hive\_id**
- uint64\_t **xgmi\_node\_id**
- uint32\_t **index**
- uint32\_t **reserved** [9]

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

- [amdsmi.h](#)



## Chapter 8

# File Documentation

### 8.1 amdsmi.h File Reference

AMD System Management Interface API.

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
```

#### Data Structures

- struct [amdsmi\\_range\\_t](#)

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

- struct [amdsmi\\_xgmi\\_info\\_t](#)
- struct [amdsmi\\_gpu\\_caps\\_t](#)
- struct [amdsmi\\_vram\\_info\\_t](#)
- struct [amdsmi\\_frequency\\_range\\_t](#)
- union [amdsmi\\_bdf\\_t](#)
- struct [amdsmi\\_power\\_cap\\_info\\_t](#)
- struct [amdsmi\\_vbios\\_info\\_t](#)
- struct [amdsmi\\_fw\\_info\\_t](#)
- struct [amdsmi\\_asic\\_info\\_t](#)
- struct [amdsmi\\_board\\_info\\_t](#)
- struct [amdsmi\\_power\\_measure\\_t](#)
- struct [amdsmi\\_clk\\_measure\\_t](#)
- struct [amdsmi\\_engine\\_usage\\_t](#)
- struct [amdsmi\\_proc\\_info\\_t](#)
- struct [amdsmi\\_counter\\_value\\_t](#)
- struct [amdsmi\\_evt\\_notification\\_data\\_t](#)
- struct [amdsmi\\_utilization\\_counter\\_t](#)

*The utilization counter data.*

- struct [amdsmi\\_retired\\_page\\_record\\_t](#)

*Reserved Memory Page Record.*

- struct [amdsmi\\_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 [amdsmi\\_frequencies\\_t](#)  
*This structure holds information about clock frequencies.*
- struct [amdsmi\\_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 [amdsmi\\_version\\_t](#)  
*This structure holds version information.*
- struct [amdsmi\\_od\\_vddc\\_point\\_t](#)  
*This structure represents a point on the frequency-voltage plane.*
- struct [amdsmi\\_freq\\_volt\\_region\\_t](#)  
*This structure holds 2 [amdsmi\\_range\\_t](#)'s, one for frequency and one for voltage. These 2 ranges indicate the range of possible values for the corresponding [amdsmi\\_od\\_vddc\\_point\\_t](#).*
- struct [amdsmi\\_od\\_volt\\_curve\\_t](#)
- struct [amdsmi\\_od\\_volt\\_freq\\_data\\_t](#)  
*This structure holds the frequency-voltage values for a device.*
- struct [amd\\_metrics\\_table\\_header\\_t](#)  
*The following structures hold the gpu metrics values for a device.*
- struct [amdsmi\\_gpu\\_metrics\\_t](#)
- struct [amdsmi\\_error\\_count\\_t](#)  
*This structure holds error counts.*
- struct [amdsmi\\_pcie\\_info\\_t](#)  
*This structure holds pcie info.*
- struct [amdsmi\\_process\\_info\\_t](#)  
*This structure contains information specific to a process.*
- union [amdsmi\\_func\\_id\\_value\\_t](#)  
*This union holds the value of an [amdsmi\\_func\\_id\\_iter\\_handle\\_t](#). The value may be a function name, or an enumerated variant value of types such as [amdsmi\\_memory\\_type\\_t](#), [amdsmi\\_temperature\\_metric\\_t](#), etc.*

## Macros

- #define **AMDSMI\_MAX\_MM\_IP\_COUNT** 8
- #define [AMDSMI\\_MAX\\_DATE\\_LENGTH](#) 32
- #define **AMDSMI\_MAX\_STRING\_LENGTH** 64
- #define **AMDSMI\_NORMAL\_STRING\_LENGTH** 32
- #define **AMDSMI\_MAX\_DEVICES** 32
- #define **AMDSMI\_MAX\_NAME** 32
- #define **AMDSMI\_MAX\_DRIVER\_VERSION\_LENGTH** 80
- #define **AMDSMI\_PRODUCT\_NAME\_LENGTH** 128
- #define **AMDSMI\_MAX\_CONTAINER\_TYPE** 2
- #define **AMDSMI\_GPU\_UUID\_SIZE** 38
- #define **AMDSMI\_TIME\_FORMAT** "%02d:%02d:%02d.%03d"
- #define **AMDSMI\_DATE\_FORMAT** "%04d-%02d-%02d:%02d:%02d.%03d"
- #define [AMDSMI\\_MAX\\_NUM\\_FREQUENCIES](#) 32  
*Guaranteed maximum possible number of supported frequencies.*
- #define [AMDSMI\\_MAX\\_FAN\\_SPEED](#) 255
- #define [AMDSMI\\_NUM\\_VOLTAGE\\_CURVE\\_POINTS](#) 3  
*The number of points that make up a voltage-frequency curve definition.*
- #define [AMDSMI\\_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 [AMDSMI\\_MAX\\_NUM\\_POWER\\_PROFILES](#) (sizeof([amdsmi\\_bit\\_field\\_t](#)) \* 8)  
*Number of possible power profiles that a system could support.*

- #define [AMDSMI\\_GPU\\_METRICS\\_API\\_FORMAT\\_VER](#) 1  
*The following structure holds the gpu metrics values for a device.*
- #define [AMDSMI\\_GPU\\_METRICS\\_API\\_CONTENT\\_VER\\_1](#) 1
- #define [AMDSMI\\_GPU\\_METRICS\\_API\\_CONTENT\\_VER\\_2](#) 2
- #define [AMDSMI\\_GPU\\_METRICS\\_API\\_CONTENT\\_VER\\_3](#) 3
- #define [AMDSMI\\_NUM\\_HBM\\_INSTANCES](#) 4
- #define [CENTRIGRADE\\_TO\\_MILLI\\_CENTIGRADE](#) 1000
- #define [AMDSMI\\_DEFAULT\\_VARIANT](#) 0xFFFFFFFFFFFFFFFF

## Typedefs

- typedef void \* [amdsmi\\_device\\_handle](#)  
*opaque handler point to underlying implementation*
- typedef void \* [amdsmi\\_socket\\_handle](#)
- typedef uint32\_t [amdsmi\\_process\\_handle](#)
- typedef uintptr\_t [amdsmi\\_event\\_handle\\_t](#)  
*Handle to performance event counter.*
- typedef uint64\_t [amdsmi\\_bit\\_field\\_t](#)  
*Bitfield used in various AMDSMI calls.*
- typedef struct amdsmi\_func\_id\_iter\_handle \* [amdsmi\\_func\\_id\\_iter\\_handle\\_t](#)  
*Opaque handle to function-support object.*

## Enumerations

- enum [amdsmi\\_init\\_flags\\_t](#) {  
[AMDSMI\\_INIT\\_ALL\\_DEVICES](#) = 0x0, [AMDSMI\\_INIT\\_AMD\\_CPUS](#) = (1 << 0), [AMDSMI\\_INIT\\_AMD\\_GPUS](#) = (1 << 1), [AMDSMI\\_INIT\\_NON\\_AMD\\_CPUS](#) = (1 << 2),  
[AMDSMI\\_INIT\\_NON\\_AMD\\_GPUS](#) = (1 << 3) }  
*Initialization flags.*
- enum [amdsmi\\_mm\\_ip\\_t](#) { [AMDSMI\\_MM\\_UVD](#), [AMDSMI\\_MM\\_VCE](#), [AMDSMI\\_MM\\_VCN](#), [AMDSMI\\_MM\\_MAX](#) }
- enum [amdsmi\\_container\\_types\\_t](#) { [CONTAINER\\_LXC](#), [CONTAINER\\_DOCKER](#) }
- enum [device\\_type\\_t](#) {  
[UNKNOWN](#) = 0, [AMD\\_GPU](#), [AMD\\_CPU](#), [NON\\_AMD\\_GPU](#),  
[NON\\_AMD\\_CPU](#) }  
*Device types detectable by AMD SMI.*
- enum [amdsmi\\_status\\_t](#) {  
[AMDSMI\\_STATUS\\_SUCCESS](#) = 0, [AMDSMI\\_STATUS\\_INVAL](#) = 1, [AMDSMI\\_STATUS\\_NOT\\_SUPPORTED](#) = 2, [AMDSMI\\_STATUS\\_NOT\\_YET\\_IMPLEMENTED](#) = 3,  
[AMDSMI\\_STATUS\\_FAIL\\_LOAD\\_MODULE](#) = 4, [AMDSMI\\_STATUS\\_FAIL\\_LOAD\\_SYMBOL](#) = 5, [AMDSMI\\_STATUS\\_DRM\\_ERROR](#) = 6, [AMDSMI\\_STATUS\\_API\\_FAILED](#) = 7,  
[AMDSMI\\_STATUS\\_TIMEOUT](#) = 8, [AMDSMI\\_STATUS\\_RETRY](#) = 9, [AMDSMI\\_STATUS\\_NO\\_PERM](#) = 10,  
[AMDSMI\\_STATUS\\_INTERRUPT](#) = 11,  
[AMDSMI\\_STATUS\\_IO](#) = 12, [AMDSMI\\_STATUS\\_ADDRESS\\_FAULT](#) = 13, [AMDSMI\\_STATUS\\_FILE\\_ERROR](#) = 14, [AMDSMI\\_STATUS\\_OUT\\_OF\\_RESOURCES](#) = 15,  
[AMDSMI\\_STATUS\\_INTERNAL\\_EXCEPTION](#) = 16, [AMDSMI\\_STATUS\\_INPUT\\_OUT\\_OF\\_BOUNDS](#) = 17,  
[AMDSMI\\_STATUS\\_INIT\\_ERROR](#) = 18, [AMDSMI\\_STATUS\\_REFCOUNT\\_OVERFLOW](#) = 19,  
[AMDSMI\\_STATUS\\_BUSY](#) = 30, [AMDSMI\\_STATUS\\_NOT\\_FOUND](#) = 31, [AMDSMI\\_STATUS\\_NOT\\_INIT](#) = 32, [AMDSMI\\_STATUS\\_NO\\_SLOT](#) = 33,  
[AMDSMI\\_STATUS\\_NO\\_DATA](#) = 40, [AMDSMI\\_STATUS\\_INSUFFICIENT\\_SIZE](#) = 41, [AMDSMI\\_STATUS\\_UNEXPECTED\\_SIZE](#) = 42, [AMDSMI\\_STATUS\\_UNEXPECTED\\_DATA](#) = 43,  
[AMDSMI\\_STATUS\\_MAP\\_ERROR](#) = 0xFFFFFFFFE, [AMDSMI\\_STATUS\\_UNKNOWN\\_ERROR](#) = 0xFFFFFFFF }





**DATA\_OUT\_LAST** = AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_5, **AMDSMI\_EVNT\_LAST** = AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_LAST }

*Event type enum. Events belonging to a particular event group [amdsmi\\_event\\_group\\_t](#) should begin enumerating at the [amdsmi\\_event\\_group\\_t](#) value for that group.*

- enum [amdsmi\\_counter\\_command\\_t](#) { **AMDSMI\_CNTR\_CMD\_START** = 0, **AMDSMI\_CNTR\_CMD\_STOP** }
- enum [amdsmi\\_evt\\_notification\\_type\\_t](#) { **AMDSMI\_EVT\_NOTIF\_VMFault** = 1, **AMDSMI\_EVT\_NOTIF\_FIRST** = AMDSMI\_EVT\_NOTIF\_VMFault, **AMDSMI\_EVT\_NOTIF\_THERMAL\_THROTTLE** = 2, **AMDSMI\_EVT\_NOTIF\_GPU\_PRE\_RESET** = 3, **AMDSMI\_EVT\_NOTIF\_GPU\_POST\_RESET** = 4, **AMDSMI\_EVT\_NOTIF\_LAST** = AMDSMI\_EVT\_NOTIF\_GPU\_POST\_RESET }
- enum [amdsmi\\_temperature\\_metric\\_t](#) { **AMDSMI\_TEMP\_CURRENT** = 0x0, **AMDSMI\_TEMP\_FIRST** = AMDSMI\_TEMP\_CURRENT, **AMDSMI\_TEMP\_MAX**, **AMDSMI\_TEMP\_MIN**, **AMDSMI\_TEMP\_MAX\_HYST**, **AMDSMI\_TEMP\_MIN\_HYST**, **AMDSMI\_TEMP\_CRITICAL**, **AMDSMI\_TEMP\_CRITICAL\_HYST**, **AMDSMI\_TEMP\_EMERGENCY**, **AMDSMI\_TEMP\_EMERGENCY\_HYST**, **AMDSMI\_TEMP\_CRIT\_MIN**, **AMDSMI\_TEMP\_CRIT\_MIN\_HYST**, **AMDSMI\_TEMP\_OFFSET**, **AMDSMI\_TEMP\_LOWEST**, **AMDSMI\_TEMP\_HIGHEST**, **AMDSMI\_TEMP\_LAST** = AMDSMI\_TEMP\_HIGHEST }

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

- enum [amdsmi\\_voltage\\_metric\\_t](#) { **AMDSMI\_VOLT\_CURRENT** = 0x0, **AMDSMI\_VOLT\_FIRST** = AMDSMI\_VOLT\_CURRENT, **AMDSMI\_VOLT\_MAX**, **AMDSMI\_VOLT\_MIN\_CRIT**, **AMDSMI\_VOLT\_MIN**, **AMDSMI\_VOLT\_MAX\_CRIT**, **AMDSMI\_VOLT\_AVERAGE**, **AMDSMI\_VOLT\_LOWEST**, **AMDSMI\_VOLT\_HIGHEST**, **AMDSMI\_VOLT\_LAST** = AMDSMI\_VOLT\_HIGHEST }

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

- enum [amdsmi\\_voltage\\_type\\_t](#) { **AMDSMI\_VOLT\_TYPE\_FIRST** = 0, **AMDSMI\_VOLT\_TYPE\_VDDGFX** = AMDSMI\_VOLT\_TYPE\_FIRST, **AMDSMI\_VOLT\_TYPE\_LAST** = AMDSMI\_VOLT\_TYPE\_VDDGFX, **AMDSMI\_VOLT\_TYPE\_INVALID** = 0xFFFFFFFF }

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

- enum [amdsmi\\_power\\_profile\\_preset\\_masks\\_t](#) { **AMDSMI\_PWR\_PROF\_PRST\_CUSTOM\_MASK** = 0x1, **AMDSMI\_PWR\_PROF\_PRST\_VIDEO\_MASK** = 0x2, **AMDSMI\_PWR\_PROF\_PRST\_POWER\_SAVING\_MASK** = 0x4, **AMDSMI\_PWR\_PROF\_PRST\_COMPUTE\_MASK** = 0x8, **AMDSMI\_PWR\_PROF\_PRST\_VR\_MASK** = 0x10, **AMDSMI\_PWR\_PROF\_PRST\_3D\_FULL\_SCR\_MASK** = 0x20, **AMDSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT** = 0x40, **AMDSMI\_PWR\_PROF\_PRST\_LAST** = AMDSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT, **AMDSMI\_PWR\_PROF\_PRST\_INVALID** = 0xFFFFFFFF }

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

- enum [amdsmi\\_gpu\\_block\\_t](#) { **AMDSMI\_GPU\_BLOCK\_INVALID** = 0x0000000000000000, **AMDSMI\_GPU\_BLOCK\_FIRST** = 0x0000000000000001, **AMDSMI\_GPU\_BLOCK\_UMC** = **AMDSMI\_GPU\_BLOCK\_FIRST**, **AMDSMI\_GPU\_BLOCK\_SDMA** = 0x0000000000000002, **AMDSMI\_GPU\_BLOCK\_GFX** = 0x0000000000000004, **AMDSMI\_GPU\_BLOCK\_MMHUB** = 0x0000000000000008, **AMDSMI\_GPU\_BLOCK\_ATHUB** = 0x0000000000000010, **AMDSMI\_GPU\_BLOCK\_PCIE\_BIF** = 0x0000000000000020, **AMDSMI\_GPU\_BLOCK\_HDP** = 0x0000000000000040, **AMDSMI\_GPU\_BLOCK\_XGMI\_WAFL** = 0x0000000000000080, **AMDSMI\_GPU\_BLOCK\_DF** = 0x0000000000000100, **AMDSMI\_GPU\_BLOCK\_SMN** = 0x0000000000000200, **AMDSMI\_GPU\_BLOCK\_SEM** = 0x0000000000000400, **AMDSMI\_GPU\_BLOCK\_MP0** = 0x0000000000000800, **AMDSMI\_GPU\_BLOCK\_MP1** = 0x0000000000001000, **AMDSMI\_GPU\_BLOCK\_FUSE** = 0x0000000000002000, **AMDSMI\_GPU\_BLOCK\_LAST** = **AMDSMI\_GPU\_BLOCK\_FUSE**, **AMDSMI\_GPU\_BLOCK\_RESERVED** = 0x8000000000000000 }

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

- enum `amdsmi_ras_err_state_t` {  
`AMDSMI_RAS_ERR_STATE_NONE` = 0, `AMDSMI_RAS_ERR_STATE_DISABLED`, `AMDSMI_RAS_ERR_STATE_PARITY`, `AMDSMI_RAS_ERR_STATE_SING_C`,  
`AMDSMI_RAS_ERR_STATE_MULT_UC`, `AMDSMI_RAS_ERR_STATE_POISON`, `AMDSMI_RAS_ERR_STATE_ENABLED`, `AMDSMI_RAS_ERR_STATE_LAST` = `AMDSMI_RAS_ERR_STATE_ENABLED`,  
`AMDSMI_RAS_ERR_STATE_INVALID` = 0xFFFFFFFF }  
*The current ECC state.*
- enum `amdsmi_memory_type_t` {  
`AMDSMI_MEM_TYPE_FIRST` = 0, `AMDSMI_MEM_TYPE_VRAM` = `AMDSMI_MEM_TYPE_FIRST`, `AMDSMI_MEM_TYPE_VIS_VRAM`, `AMDSMI_MEM_TYPE_GTT`,  
`AMDSMI_MEM_TYPE_LAST` = `AMDSMI_MEM_TYPE_GTT` }  
*Types of memory.*
- enum `amdsmi_freq_ind_t` { `AMDSMI_FREQ_IND_MIN` = 0, `AMDSMI_FREQ_IND_MAX` = 1, `AMDSMI_FREQ_IND_INVALID` = 0xFFFFFFFF }  
*The values of this enum are used as frequency identifiers.*
- enum `amdsmi_xgmi_status_t` { `AMDSMI_XGMI_STATUS_NO_ERRORS` = 0, `AMDSMI_XGMI_STATUS_ERROR`, `AMDSMI_XGMI_STATUS_MULTIPLE_ERRORS` }  
*XGMI Status.*
- enum `amdsmi_memory_page_status_t` { `AMDSMI_MEM_PAGE_STATUS_RESERVED` = 0, `AMDSMI_MEM_PAGE_STATUS_PENDING`, `AMDSMI_MEM_PAGE_STATUS_UNRESERVABLE` }  
*Reserved Memory Page States.*
- enum `AMDSMI_IO_LINK_TYPE` {  
`AMDSMI_IOLINK_TYPE_UNDEFINED` = 0, `AMDSMI_IOLINK_TYPE_PCIEXPRESS` = 1, `AMDSMI_IOLINK_TYPE_XGMI` = 2, `AMDSMI_IOLINK_TYPE_NUMIOLINKTYPES`,  
`AMDSMI_IOLINK_TYPE_SIZE` = 0xFFFFFFFF }  
*Types for IO Link.*
- enum `AMDSMI_UTILIZATION_COUNTER_TYPE` { `AMDSMI_UTILIZATION_COUNTER_FIRST` = 0, `AMDSMI_COARSE_GRAIN_GFX_ACTIVITY` = `AMDSMI_UTILIZATION_COUNTER_FIRST`, `AMDSMI_COARSE_GRAIN_MEM_ACTIVITY`, `AMDSMI_UTILIZATION_COUNTER_LAST` = `AMDSMI_COARSE_GRAIN_MEM_ACTIVITY` }  
*The utilization counter type.*

## Functions

- `amdsmi_status_t amdsmi_init` (uint64\_t init\_flags)  
*Initialize the AMD SMI library.*
- `amdsmi_status_t amdsmi_shut_down` (void)  
*Shutdown the AMD SMI library.*
- `amdsmi_status_t amdsmi_get_socket_handles` (uint32\_t \*socket\_count, amdsmi\_socket\_handle \*socket\_handles)  
*Get the list of socket handles in the system.*
- `amdsmi_status_t amdsmi_get_socket_info` (amdsmi\_socket\_handle socket\_handle, char \*name, size\_t len)  
*Get information about the given socket.*
- `amdsmi_status_t amdsmi_get_device_handles` (amdsmi\_socket\_handle socket\_handle, uint32\_t \*device\_count, amdsmi\_device\_handle \*device\_handles)  
*Get the list of the device handles associated to a socket.*
- `amdsmi_status_t amdsmi_get_device_type` (amdsmi\_device\_handle device\_handle, device\_type\_t \*device\_type)  
*Get the device type of the device\_handle.*
- `amdsmi_status_t amdsmi_get_device_handle_from_bdf` (amdsmi\_bdf\_t bdf, amdsmi\_device\_handle \*device\_handle)  
*Get device handle with the matching bdf.*
- `amdsmi_status_t amdsmi_dev_get_id` (amdsmi\_device\_handle device\_handle, uint16\_t \*id)

*Get the device id associated with the device with provided device handler.*

- `amdsmi_status_t amdsmi_dev_get_vendor_name (amdsmi_device_handle device_handle, char *name, size_t len)`

*Get the name string for a give vendor ID.*

- `amdsmi_status_t amdsmi_dev_get_vram_vendor (amdsmi_device_handle device_handle, char *brand, uint32_t len)`

*Get the vram vendor string of a device.*

- `amdsmi_status_t amdsmi_dev_get_subsystem_id (amdsmi_device_handle device_handle, uint16_t *id)`

*Get the subsystem device id associated with the device with provided device handle.*

- `amdsmi_status_t amdsmi_dev_get_subsystem_name (amdsmi_device_handle device_handle, char *name, size_t len)`

*Get the name string for the device subsystem.*

- `amdsmi_status_t amdsmi_dev_get_drm_render_minor (amdsmi_device_handle device_handle, uint32_t *minor)`

*Get the drm minor number associated with this device.*

- `amdsmi_status_t amdsmi_dev_get_pci_bandwidth (amdsmi_device_handle device_handle, amdsmi_pcie_bandwidth_t *bandwidth)`

*Get the list of possible PCIe bandwidths that are available.*

- `amdsmi_status_t amdsmi_dev_get_pci_id (amdsmi_device_handle device_handle, uint64_t *bdfid)`

*Get the unique PCI device identifier associated for a device.*

- `amdsmi_status_t amdsmi_topo_get_numa_affinity (amdsmi_device_handle device_handle, uint32_t *numa_node)`

*Get the NUMA node associated with a device.*

- `amdsmi_status_t amdsmi_dev_get_pci_throughput (amdsmi_device_handle device_handle, uint64_t *sent, uint64_t *received, uint64_t *max_pkt_sz)`

*Get PCIe traffic information.*

- `amdsmi_status_t amdsmi_dev_get_pci_replay_counter (amdsmi_device_handle device_handle, uint64_t *counter)`

*Get PCIe replay counter.*

- `amdsmi_status_t amdsmi_dev_set_pci_bandwidth (amdsmi_device_handle device_handle, uint64_t bw_bitmask)`

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

- `amdsmi_status_t amdsmi_dev_get_power_ave (amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t *power)`

*Get the average power consumption of a device.*

- `amdsmi_status_t amdsmi_dev_get_energy_count (amdsmi_device_handle device_handle, uint64_t *power, float *counter_resolution, uint64_t *timestamp)`

*Get the energy accumulator counter of the device with provided device handle.*

- `amdsmi_status_t amdsmi_dev_set_power_cap (amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t cap)`

*Set the maximum gpu power cap value.*

- `amdsmi_status_t amdsmi_dev_set_power_profile (amdsmi_device_handle device_handle, uint32_t reserved, amdsmi_power_profile_preset_masks_t profile)`

*Set the power performance profile.*

- `amdsmi_status_t amdsmi_dev_get_memory_total (amdsmi_device_handle device_handle, amdsmi_memory_type_t mem_type, uint64_t *total)`

*Get the total amount of memory that exists.*

- `amdsmi_status_t amdsmi_dev_get_memory_usage (amdsmi_device_handle device_handle, amdsmi_memory_type_t mem_type, uint64_t *used)`

*Get the current memory usage.*

- `amdsmi_status_t amdsmi_get_bad_page_info (amdsmi_device_handle device_handle, uint32_t *num_pages, amdsmi_retired_page_record_t *info)`

The first call to this API returns the number of bad pages which should be used to allocate the buffer that should contain the bad page records.

- `amdsmi_status_t amdsmi_get_ras_block_features_enabled (amdsmi_device_handle device_handle, amdsmi_gpu_block_t block, amdsmi_ras_err_state_t *state)`

Returns if RAS features are enabled or disabled for given block.

- `amdsmi_status_t amdsmi_dev_get_memory_busy_percent (amdsmi_device_handle device_handle, uint32_t *busy_percent)`

Get percentage of time any device memory is being used.

- `amdsmi_status_t amdsmi_dev_get_memory_reserved_pages (amdsmi_device_handle device_handle, uint32_t *num_pages, amdsmi_retired_page_record_t *records)`

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

- `amdsmi_status_t amdsmi_dev_get_fan_rpms (amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t *speed)`

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

- `amdsmi_status_t amdsmi_dev_get_fan_speed (amdsmi_device_handle device_handle, uint32_t sensor_ind, int64_t *speed)`

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

- `amdsmi_status_t amdsmi_dev_get_fan_speed_max (amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t *max_speed)`

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

- `amdsmi_status_t amdsmi_dev_get_temp_metric (amdsmi_device_handle device_handle, amdsmi_temperature_type_t sensor_type, amdsmi_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.

- `amdsmi_status_t amdsmi_dev_get_volt_metric (amdsmi_device_handle device_handle, amdsmi_voltage_type_t sensor_type, amdsmi_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.

- `amdsmi_status_t amdsmi_dev_reset_fan (amdsmi_device_handle device_handle, uint32_t sensor_ind)`

Reset the fan to automatic driver control.

- `amdsmi_status_t amdsmi_dev_set_fan_speed (amdsmi_device_handle device_handle, uint32_t sensor_ind, uint64_t speed)`

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

- `amdsmi_status_t amdsmi_dev_get_busy_percent (amdsmi_device_handle device_handle, uint32_t *busy_percent)`

Get percentage of time device is busy doing any processing.

- `amdsmi_status_t amdsmi_get_utilization_count (amdsmi_device_handle device_handle, amdsmi_utilization_counter_t utilization_counters[], uint32_t count, uint64_t *timestamp)`

Get coarse grain utilization counter of the specified device.

- `amdsmi_status_t amdsmi_get_pcie_link_status (amdsmi_device_handle device_handle, amdsmi_pcie_info_t *info)`

Get current PCIE info of the device with provided device handle.

- `amdsmi_status_t amdsmi_get_pcie_link_caps (amdsmi_device_handle device_handle, amdsmi_pcie_info_t *info)`

Get max PCIe capabilities of the device with provided device handle.

- `amdsmi_status_t amdsmi_dev_get_perf_level (amdsmi_device_handle device_handle, amdsmi_dev_perf_level_t *perf)`

Get the performance level of the device.

- `amdsmi_status_t amdsmi_set_perf_determinism_mode (amdsmi_device_handle device_handle, uint64_t clkvalue)`

Enter performance determinism mode with provided device handle.

- `amdsmi_status_t amdsmi_dev_get_overdrive_level (amdsmi_device_handle device_handle, uint32_t *od)`

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

- `amdsmi_status_t amdsmi_dev_get_gpu_clk_freq` (`amdsmi_device_handle` device\_handle, `amdsmi_clk_type_t` clk\_type, `amdsmi_frequencies_t` \*f)  
*Get the list of possible system clock speeds of device for a specified clock type.*
- `amdsmi_status_t amdsmi_dev_reset_gpu` (`amdsmi_device_handle` device\_handle)  
*Reset the gpu associated with the device with provided device handle.*
- `amdsmi_status_t amdsmi_dev_get_od_volt_info` (`amdsmi_device_handle` device\_handle, `amdsmi_od_volt_freq_data_t` \*odv)  
*This function retrieves the voltage/frequency curve information.*
- `amdsmi_status_t amdsmi_dev_get_gpu_metrics_info` (`amdsmi_device_handle` device\_handle, `amdsmi_gpu_metrics_t` \*pgpu\_metrics)  
*This function retrieves the gpu metrics information.*
- `amdsmi_status_t amdsmi_dev_set_clk_range` (`amdsmi_device_handle` device\_handle, `uint64_t` minclkvalue, `uint64_t` maxclkvalue, `amdsmi_clk_type_t` clkType)  
*This function sets the clock range information.*
- `amdsmi_status_t amdsmi_dev_set_od_clk_info` (`amdsmi_device_handle` device\_handle, `amdsmi_freq_index_t` level, `uint64_t` clkvalue, `amdsmi_clk_type_t` clkType)  
*This function sets the clock frequency information.*
- `amdsmi_status_t amdsmi_dev_set_od_volt_info` (`amdsmi_device_handle` device\_handle, `uint32_t` vpoint, `uint64_t` clkvalue, `uint64_t` voltvalue)  
*This function sets 1 of the 3 voltage curve points.*
- `amdsmi_status_t amdsmi_dev_get_od_volt_curve_regions` (`amdsmi_device_handle` device\_handle, `uint32_t` \*num\_regions, `amdsmi_freq_volt_region_t` \*buffer)  
*This function will retrieve the current valid regions in the frequency/voltage space.*
- `amdsmi_status_t amdsmi_dev_get_power_profile_presets` (`amdsmi_device_handle` device\_handle, `uint32_t` sensor\_ind, `amdsmi_power_profile_status_t` \*status)  
*Get the list of available preset power profiles and an indication of which profile is currently active.*
- `amdsmi_status_t amdsmi_dev_set_perf_level` (`amdsmi_device_handle` device\_handle, `amdsmi_dev_perf_level_t` perf\_lvl)  
*Set the PowerPlay performance level associated with the device with provided device handle with the provided value.*
- `amdsmi_status_t amdsmi_dev_set_perf_level_v1` (`amdsmi_device_handle` device\_handle, `amdsmi_dev_perf_level_t` perf\_lvl)  
*Set the PowerPlay performance level associated with the device with provided device handle with the provided value.*
- `amdsmi_status_t amdsmi_dev_set_overdrive_level` (`amdsmi_device_handle` device\_handle, `uint32_t` od)  
*Set the overdrive percent associated with the device with provided device handle with the provided value. See details for WARNING.*
- `amdsmi_status_t amdsmi_dev_set_overdrive_level_v1` (`amdsmi_device_handle` device\_handle, `uint32_t` od)  
*Set the overdrive percent associated with the device with provided device handle with the provided value. See details for WARNING.*
- `amdsmi_status_t amdsmi_dev_set_clk_freq` (`amdsmi_device_handle` device\_handle, `amdsmi_clk_type_t` clk\_type, `uint64_t` freq\_bitmask)  
*Control the set of allowed frequencies that can be used for the specified clock.*
- `amdsmi_status_t amdsmi_get_version` (`amdsmi_version_t` \*version)  
*Get the build version information for the currently running build of AMDSMI.*
- `amdsmi_status_t amdsmi_get_version_str` (`amdsmi_sw_component_t` component, `char` \*ver\_str, `uint32_t` len)  
*Get the driver version string for the current system.*
- `amdsmi_status_t amdsmi_dev_get_ecc_count` (`amdsmi_device_handle` device\_handle, `amdsmi_gpu_block_t` block, `amdsmi_error_count_t` \*ec)  
*Retrieve the error counts for a GPU block.*
- `amdsmi_status_t amdsmi_dev_get_ecc_enabled` (`amdsmi_device_handle` device\_handle, `uint64_t` \*enabled\_blocks)  
*Retrieve the enabled ECC bit-mask.*

- `amdsmi_status_t amdsmi_dev_get_ecc_status` (`amdsmi_device_handle` device\_handle, `amdsmi_gpu_block_t` block, `amdsmi_ras_err_state_t` \*state)  
*Retrieve the ECC status for a GPU block.*
- `amdsmi_status_t amdsmi_status_string` (`amdsmi_status_t` status, `const char **`status\_string)  
*Get a description of a provided AMDSMI error status.*
- `amdsmi_status_t amdsmi_dev_counter_group_supported` (`amdsmi_device_handle` device\_handle, `amdsmi_event_group_t` group)  
*Tell if an event group is supported by a given device.*
- `amdsmi_status_t amdsmi_dev_create_counter` (`amdsmi_device_handle` device\_handle, `amdsmi_event_type_t` type, `amdsmi_event_handle_t` \*evnt\_handle)  
*Create a performance counter object.*
- `amdsmi_status_t amdsmi_dev_destroy_counter` (`amdsmi_event_handle_t` evnt\_handle)  
*Deallocate a performance counter object.*
- `amdsmi_status_t amdsmi_control_counter` (`amdsmi_event_handle_t` evt\_handle, `amdsmi_counter_command_t` cmd, `void *cmd_args`)  
*Issue performance counter control commands.*
- `amdsmi_status_t amdsmi_read_counter` (`amdsmi_event_handle_t` evt\_handle, `amdsmi_counter_value_t` \*value)  
*Read the current value of a performance counter.*
- `amdsmi_status_t amdsmi_counter_get_available_counters` (`amdsmi_device_handle` device\_handle, `amdsmi_event_group_t` grp, `uint32_t` \*available)  
*Get the number of currently available counters.*
- `amdsmi_status_t amdsmi_get_compute_process_info` (`amdsmi_process_info_t` \*procs, `uint32_t` \*num\_items)  
*Get process information about processes currently using GPU.*
- `amdsmi_status_t amdsmi_get_compute_process_info_by_pid` (`uint32_t` pid, `amdsmi_process_info_t` \*proc)  
*Get process information about a specific process.*
- `amdsmi_status_t amdsmi_get_compute_process_gpus` (`uint32_t` pid, `uint32_t` \*dv\_indices, `uint32_t` \*num\_devices)  
*Get the device indices currently being used by a process.*
- `amdsmi_status_t amdsmi_dev_xgmi_error_status` (`amdsmi_device_handle` device\_handle, `amdsmi_xgmi_status_t` \*status)  
*Retrieve the XGMI error status for a device.*
- `amdsmi_status_t amdsmi_dev_reset_xgmi_error` (`amdsmi_device_handle` device\_handle)  
*Reset the XGMI error status for a device.*
- `amdsmi_status_t amdsmi_topo_get_numa_node_number` (`amdsmi_device_handle` device\_handle, `uint32_t` \*numa\_node)  
*Retrieve the NUMA CPU node number for a device.*
- `amdsmi_status_t amdsmi_topo_get_link_weight` (`amdsmi_device_handle` device\_handle\_src, `amdsmi_device_handle` device\_handle\_dst, `uint64_t` \*weight)  
*Retrieve the weight for a connection between 2 GPUs.*
- `amdsmi_status_t amdsmi_get_minmax_bandwidth` (`amdsmi_device_handle` device\_handle\_src, `amdsmi_device_handle` device\_handle\_dst, `uint64_t` \*min\_bandwidth, `uint64_t` \*max\_bandwidth)  
*Retrieve minimal and maximal io link bandwidth between 2 GPUs.*
- `amdsmi_status_t amdsmi_topo_get_link_type` (`amdsmi_device_handle` device\_handle\_src, `amdsmi_device_handle` device\_handle\_dst, `uint64_t` \*hops, `AMDSMI_IO_LINK_TYPE` \*type)  
*Retrieve the hops and the connection type between 2 GPUs.*
- `amdsmi_status_t amdsmi_is_P2P_accessible` (`amdsmi_device_handle` device\_handle\_src, `amdsmi_device_handle` device\_handle\_dst, `bool` \*accessible)  
*Return P2P availability status between 2 GPUs.*
- `amdsmi_status_t amdsmi_dev_open_supported_func_iterator` (`amdsmi_device_handle` device\_handle, `amdsmi_func_id_iter_handle_t` \*handle)  
*Get a function name iterator of supported AMDSMI functions for a device.*

- `amdsmi_status_t amdsmi_dev_open_supported_variant_iterator` (`amdsmi_func_id_iter_handle_t` obj\_h, `amdsmi_func_id_iter_handle_t` \*var\_iter)  
*Get a variant iterator for a given handle.*
- `amdsmi_status_t amdsmi_next_func_iter` (`amdsmi_func_id_iter_handle_t` handle)  
*Advance a function identifier iterator.*
- `amdsmi_status_t amdsmi_dev_close_supported_func_iterator` (`amdsmi_func_id_iter_handle_t` \*handle)  
*Close a variant iterator handle.*
- `amdsmi_status_t amdsmi_get_func_iter_value` (`amdsmi_func_id_iter_handle_t` handle, `amdsmi_func_id_value_t` \*value)  
*Get the value associated with a function/variant iterator.*
- `amdsmi_status_t amdsmi_init_event_notification` (`amdsmi_device_handle` device\_handle)  
*Prepare to collect event notifications for a GPU.*
- `amdsmi_status_t amdsmi_set_event_notification_mask` (`amdsmi_device_handle` device\_handle, `uint64_t` mask)  
*Specify which events to collect for a device.*
- `amdsmi_status_t amdsmi_get_event_notification` (`int` timeout\_ms, `uint32_t` \*num\_elem, `amdsmi_evt_notification_data_t` \*data)  
*Collect event notifications, waiting a specified amount of time.*
- `amdsmi_status_t amdsmi_stop_event_notification` (`amdsmi_device_handle` device\_handle)  
*Close any file handles and free any resources used by event notification for a GPU.*
- `amdsmi_status_t amdsmi_get_device_bdf` (`amdsmi_device_handle` device\_handle, `amdsmi_bdf_t` \*bdf)  
*Returns BDF of the given device.*
- `amdsmi_status_t amdsmi_get_device_uuid` (`amdsmi_device_handle` device\_handle, `unsigned int` \*uuid\_length, `char` \*uuid)  
*Returns the UUID of the device.*
- `amdsmi_status_t amdsmi_get_driver_version` (`amdsmi_device_handle` device\_handle, `int` \*length, `char` \*version)  
*Returns the driver version information.*
- `amdsmi_status_t amdsmi_get_asic_info` (`amdsmi_device_handle` device\_handle, `amdsmi_asic_info_t` \*info)  
*Returns the ASIC information for the device.*
- `amdsmi_status_t amdsmi_get_board_info` (`amdsmi_device_handle` device\_handle, `amdsmi_board_info_t` \*info)  
*Returns the board part number and board information for the requested device.*
- `amdsmi_status_t amdsmi_get_power_cap_info` (`amdsmi_device_handle` device\_handle, `uint32_t` sensor\_ind, `amdsmi_power_cap_info_t` \*info)  
*Returns the power caps as currently configured in the system.*
- `amdsmi_status_t amdsmi_get_xgmi_info` (`amdsmi_device_handle` device\_handle, `amdsmi_xgmi_info_t` \*info)  
*Returns XGMI information for the GPU.*
- `amdsmi_status_t amdsmi_get_caps_info` (`amdsmi_device_handle` device\_handle, `amdsmi_gpu_caps_t` \*info)  
*Returns the device capabilities as currently configured in the system.*
- `amdsmi_status_t amdsmi_get_fw_info` (`amdsmi_device_handle` device\_handle, `amdsmi_fw_info_t` \*info)  
*Returns the firmware versions running on the device.*
- `amdsmi_status_t amdsmi_get_vbios_info` (`amdsmi_device_handle` device\_handle, `amdsmi_vbios_info_t` \*info)  
*Returns the static information for the vBIOS on the device.*
- `amdsmi_status_t amdsmi_get_gpu_activity` (`amdsmi_device_handle` device\_handle, `amdsmi_engine_usage_t` \*info)  
*Returns the current usage of the GPU engines (GFX, MM and MEM). Each usage is reported as a percentage from 0-100%.*
- `amdsmi_status_t amdsmi_get_power_measure` (`amdsmi_device_handle` device\_handle, `amdsmi_power_measure_t` \*info)



*Returns the current power and voltage of the GPU. The voltage is in units of mV and the power in units of W.*

- `amdsmi_status_t amdsmi_get_clock_measure (amdsmi_device_handle device_handle, amdsmi_clk_type_t clk_type, amdsmi_clk_measure_t *info)`

*Returns the measurements of the clocks in the GPU for the GFX and multimedia engines and Memory. This call reports the averages over 1s in MHz.*

- `amdsmi_status_t amdsmi_get_vram_usage (amdsmi_device_handle device_handle, amdsmi_vram_info_t *info)`

*Returns the VRAM usage (both total and used memory) in MegaBytes.*

- `amdsmi_status_t amdsmi_get_target_frequency_range (amdsmi_device_handle device_handle, amdsmi_clk_type_t clk_type, amdsmi_frequency_range_t *range)`

*Returns current and supported frequency range for the specified clock type.*

- `amdsmi_status_t amdsmi_get_process_list (amdsmi_device_handle device_handle, amdsmi_process_handle *list, uint32_t *max_processes)`

*Returns the list of processes running on a given GPU including itself.*

- `amdsmi_status_t amdsmi_get_process_info (amdsmi_device_handle device_handle, amdsmi_process_handle process, amdsmi_proc_info_t *info)`

*Returns the process information of a given process. Engine usage show how much time the process spend using these engines in ns.*

- `amdsmi_status_t amdsmi_get_ecc_error_count (amdsmi_device_handle device_handle, amdsmi_error_count_t *ec)`

*Returns the number of ECC errors (correctable and uncorrectable) in the given GPU.*

### 8.1.1 Detailed Description

AMD System Management Interface API.

### 8.1.2 Macro Definition Documentation

#### 8.1.2.1 `#define AMDSMI_MAX_DATE_LENGTH 32`

YYYY-MM-DD:HH:MM:SS.MSC

#### 8.1.2.2 `#define AMDSMI_MAX_FAN_SPEED 255`

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

#### 8.1.2.3 `#define AMDSMI_EVENT_MASK_FROM_INDEX( i ) (1ULL << ((i) - 1))`

Macro to generate event bitmask from event id

#### 8.1.2.4 `#define AMDSMI_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 typedef uintptr\_t amdsmi\_event\_handle\_t

Handle to performance event counter.

Event counter types

### 8.1.4 Enumeration Type Documentation

#### 8.1.4.1 enum amdsmi\_init\_flags\_t

Initialization flags.

Initialization flags may be OR'd together and passed to [amdsmi\\_init\(\)](#).

#### 8.1.4.2 enum amdsmi\_status\_t

Error codes returned by amdsmi functions.

Enumerator

**AMDSMI\_STATUS\_SUCCESS** Call succeeded.

**AMDSMI\_STATUS\_INVALID** Invalid parameters.

**AMDSMI\_STATUS\_NOT\_SUPPORTED** Command not supported.

**AMDSMI\_STATUS\_NOT\_YET\_IMPLEMENTED** Not implemented yet.

**AMDSMI\_STATUS\_FAIL\_LOAD\_MODULE** Fail to load lib.

**AMDSMI\_STATUS\_FAIL\_LOAD\_SYMBOL** Fail to load symbol.

**AMDSMI\_STATUS\_DRM\_ERROR** Error when call libdrm.

**AMDSMI\_STATUS\_API\_FAILED** API call failed.

**AMDSMI\_STATUS\_TIMEOUT** Timeout in API call.

**AMDSMI\_STATUS\_RETRY** Retry operation.

**AMDSMI\_STATUS\_NO\_PERM** Permission Denied.

**AMDSMI\_STATUS\_INTERRUPT** An interrupt occurred during execution of function.

**AMDSMI\_STATUS\_IO** I/O Error.

**AMDSMI\_STATUS\_ADDRESS\_FAULT** Bad address.

**AMDSMI\_STATUS\_FILE\_ERROR** Problem accessing a file.

**AMDSMI\_STATUS\_OUT\_OF\_RESOURCES** Not enough memory.

**AMDSMI\_STATUS\_INTERNAL\_EXCEPTION** An internal exception was caught.

**AMDSMI\_STATUS\_INPUT\_OUT\_OF\_BOUNDS** The provided input is out of allowable or safe range.

**AMDSMI\_STATUS\_INIT\_ERROR** An error occurred when initializing internal data structures.

**AMDSMI\_STATUS\_REFCOUNT\_OVERFLOW** An internal reference counter exceeded INT32\_MAX.

**AMDSMI\_STATUS\_BUSY** Device busy.

**AMDSMI\_STATUS\_NOT\_FOUND** Device Not found.

**AMDSMI\_STATUS\_NOT\_INIT** Device not initialized.

**AMDSMI\_STATUS\_NO\_SLOT** No more free slot.

**AMDSMI\_STATUS\_NO\_DATA** No data was found for a given input.

**AMDSMI\_STATUS\_INSUFFICIENT\_SIZE** Not enough resources were available for the operation.

**AMDSMI\_STATUS\_UNEXPECTED\_SIZE** An unexpected amount of data was read.

**AMDSMI\_STATUS\_UNEXPECTED\_DATA** The data read or provided to function is not what was expected.

  

**AMDSMI\_STATUS\_MAP\_ERROR** The internal library error did not map to a status code.

**AMDSMI\_STATUS\_UNKNOWN\_ERROR** An unknown error occurred.

#### 8.1.4.3 enum amdsmi\_clk\_type\_t

Clock types

Enumerator

**CLK\_TYPE\_SYS** System clock.

**CLK\_TYPE\_DF** running on a separate clock) Data Fabric clock (for ASICs

**CLK\_TYPE\_DCEF** Display Controller Engine clock.

#### 8.1.4.4 enum amdsmi\_dev\_perf\_level\_t

PowerPlay performance levels.

Enumerator

**AMD\_SMI\_DEV\_PERF\_LEVEL\_AUTO** Performance level is "auto".

**AMD\_SMI\_DEV\_PERF\_LEVEL\_LOW** regardless of workload Keep PowerPlay levels "low",

**AMD\_SMI\_DEV\_PERF\_LEVEL\_HIGH** regardless of workload Keep PowerPlay levels "high",

**AMD\_SMI\_DEV\_PERF\_LEVEL\_MANUAL** setting the AMD\_SMI\_CLK\_TYPE\_SYS speed Only use values defined by manually

**AMD\_SMI\_DEV\_PERF\_LEVEL\_STABLE\_STD** clocks Stable power state with profiling

**AMD\_SMI\_DEV\_PERF\_LEVEL\_STABLE\_PEAK** Stable power state with peak clocks.

**AMD\_SMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_MCLK** memory clock Stable power state with minimum

**AMD\_SMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_SCLK** system clock Stable power state with minimum

**AMD\_SMI\_DEV\_PERF\_LEVEL\_DETERMINISM** Performance determinism state.

**AMD\_SMI\_DEV\_PERF\_LEVEL\_UNKNOWN** Unknown performance level.

#### 8.1.4.5 enum amdsmi\_sw\_component\_t

Available clock types.

Software components

Enumerator

**AMD\_SMI\_SW\_COMP\_DRIVER** Driver.

#### 8.1.4.6 enum amdsmi\_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

**AMD\_SMI\_EVNT\_GRP\_XGMI** Data Fabric (XGMI) related events.

**AMD\_SMI\_EVNT\_GRP\_XGMI\_DATA\_OUT** XGMI Outbound data.

## 8.1.4.7 enum amdsmi\_event\_type\_t

Event type enum. Events belonging to a particular event group [amdsmi\\_event\\_group\\_t](#) should begin enumerating at the [amdsmi\\_event\\_group\\_t](#) value for that group.

Event types

Enumerator

**AMDSMI\_EVNT\_XGMI\_0\_NOP\_TX** NOPs sent to neighbor 0.

**AMDSMI\_EVNT\_XGMI\_0\_REQUEST\_TX** neighbor 0 Outgoing requests to

**AMDSMI\_EVNT\_XGMI\_0\_RESPONSE\_TX** neighbor 0 Outgoing responses to

**AMDSMI\_EVNT\_XGMI\_0\_BEATS\_TX** Data beats sent to neighbor 0; Each beat represents 32 bytes.

XGMI throughput can be calculated by multiplying a BEATS event such as [AMDSMI\\_EVNT\\_XGMI\\_0\\_BEATS\\_TX](#) by 32 and dividing by the time for which event collection occurred, [amdsmi\\_counter\\_value\\_t.time\\_running](#) (which is in nanoseconds). To get bytes per second, multiply this value by  $10^9$ .

Throughput = BEATS/time\_running \*  $10^9$  (bytes/second)

**AMDSMI\_EVNT\_XGMI\_1\_NOP\_TX** NOPs sent to neighbor 1.

**AMDSMI\_EVNT\_XGMI\_1\_REQUEST\_TX** neighbor 1 Outgoing requests to

**AMDSMI\_EVNT\_XGMI\_1\_RESPONSE\_TX** neighbor 1 Outgoing responses to

**AMDSMI\_EVNT\_XGMI\_1\_BEATS\_TX** Data beats sent to neighbor 1; Each beat represents 32 bytes

**AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_1** Outbound beats to neighbor 1.

**AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_2** Outbound beats to neighbor 2.

**AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_3** Outbound beats to neighbor 3.

**AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_4** Outbound beats to neighbor 4.

**AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_5** Outbound beats to neighbor 5.

## 8.1.4.8 enum amdsmi\_counter\_command\_t

Event counter commands

Enumerator

**AMDSMI\_CNTR\_CMD\_START** Start the counter.

**AMDSMI\_CNTR\_CMD\_STOP** be used before reading. Stop the counter; note that this should not

## 8.1.4.9 enum amdsmi\_evt\_notification\_type\_t

Event notification event types

Enumerator

**AMDSMI\_EVT\_NOTIF\_VMFAULT** VM page fault.

#### 8.1.4.10 enum amdsmi\_temperature\_metric\_t

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

##### Enumerator

- AMDSMI\_TEMP\_CURRENT** Temperature current value.
- AMDSMI\_TEMP\_MAX** Temperature max value.
- AMDSMI\_TEMP\_MIN** Temperature min value.
- AMDSMI\_TEMP\_MAX\_HYST** Temperature hysteresis value for max limit. (This is an absolute temperature, not a delta).
- AMDSMI\_TEMP\_MIN\_HYST** Temperature hysteresis value for min limit. (This is an absolute temperature, not a delta).
- AMDSMI\_TEMP\_CRITICAL** greater than corresponding temp\_max values. Temperature critical max value, typically
- AMDSMI\_TEMP\_CRITICAL\_HYST** Temperature hysteresis value for critical limit. (This is an absolute temperature, not a delta).
- AMDSMI\_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.
- AMDSMI\_TEMP\_EMERGENCY\_HYST** Temperature hysteresis value for emergency limit. (This is an absolute temperature, not a delta).
- AMDSMI\_TEMP\_CRIT\_MIN** Temperature critical min value, typically lower than corresponding temperature minimum values.
- AMDSMI\_TEMP\_CRIT\_MIN\_HYST** Temperature hysteresis value for critical minimum limit. (This is an absolute temperature, not a delta).
- AMDSMI\_TEMP\_OFFSET** Temperature offset which is added to the
- AMDSMI\_TEMP\_LOWEST** temperature reading by the chip. Historical minimum temperature.
- AMDSMI\_TEMP\_HIGHEST** Historical maximum temperature.

#### 8.1.4.11 enum amdsmi\_voltage\_metric\_t

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

##### Enumerator

- AMDSMI\_VOLT\_CURRENT** Voltage current value.
- AMDSMI\_VOLT\_MAX** Voltage max value.
- AMDSMI\_VOLT\_MIN\_CRIT** Voltage critical min value.
- AMDSMI\_VOLT\_MIN** Voltage min value.
- AMDSMI\_VOLT\_MAX\_CRIT** Voltage critical max value.
- AMDSMI\_VOLT\_AVERAGE** Average voltage.
- AMDSMI\_VOLT\_LOWEST** Historical minimum voltage.
- AMDSMI\_VOLT\_HIGHEST** Historical maximum voltage.

## 8.1.4.12 enum amdsmi\_voltage\_type\_t

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

Enumerator

**AMDSMI\_VOLT\_TYPE\_VDDGFX** voltage Vddgfx GPU  
**AMDSMI\_VOLT\_TYPE\_INVALID** Invalid type.

## 8.1.4.13 enum amdsmi\_power\_profile\_preset\_masks\_t

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

Enumerator

**AMDSMI\_PWR\_PROF\_PRST\_CUSTOM\_MASK** Custom Power Profile.  
**AMDSMI\_PWR\_PROF\_PRST\_VIDEO\_MASK** Video Power Profile.  
**AMDSMI\_PWR\_PROF\_PRST\_POWER\_SAVING\_MASK** Power Saving Profile.  
**AMDSMI\_PWR\_PROF\_PRST\_COMPUTE\_MASK** Compute Saving Profile.  
**AMDSMI\_PWR\_PROF\_PRST\_VR\_MASK** VR Power Profile. 3D Full Screen Power Profile  
**AMDSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT** Default Boot Up Profile.  
**AMDSMI\_PWR\_PROF\_PRST\_LAST** Invalid power profile.

## 8.1.4.14 enum amdsmi\_gpu\_block\_t

This enum is used to identify different GPU blocks.

Enumerator

**AMDSMI\_GPU\_BLOCK\_INVALID** invalid block Used to indicate an  
**AMDSMI\_GPU\_BLOCK\_UMC** UMC block.  
**AMDSMI\_GPU\_BLOCK\_SDMA** SDMA block.  
**AMDSMI\_GPU\_BLOCK\_GFX** GFX block.  
**AMDSMI\_GPU\_BLOCK\_MMHUB** MMHUB block.  
**AMDSMI\_GPU\_BLOCK\_ATHUB** ATHUB block.  
**AMDSMI\_GPU\_BLOCK\_PCIE\_BIF** PCIE\_BIF block.  
**AMDSMI\_GPU\_BLOCK\_HDP** HDP block.  
**AMDSMI\_GPU\_BLOCK\_XGMI\_WAFL** XGMI block.  
**AMDSMI\_GPU\_BLOCK\_DF** DF block.  
**AMDSMI\_GPU\_BLOCK\_SMN** SMN block.  
**AMDSMI\_GPU\_BLOCK\_SEM** SEM block.  
**AMDSMI\_GPU\_BLOCK\_MP0** MP0 block.  
**AMDSMI\_GPU\_BLOCK\_MP1** MP1 block.  
**AMDSMI\_GPU\_BLOCK\_FUSE** Fuse block.  
**AMDSMI\_GPU\_BLOCK\_LAST** for supported blocks The highest bit position

## 8.1.4.15 enum amdsmi\_ras\_err\_state\_t

The current ECC state.

Enumerator

**AMDSMI\_RAS\_ERR\_STATE\_NONE** No current errors.  
**AMDSMI\_RAS\_ERR\_STATE\_DISABLED** ECC is disabled.  
**AMDSMI\_RAS\_ERR\_STATE\_PARITY** ECC errors present, but type unknown.  
**AMDSMI\_RAS\_ERR\_STATE\_SING\_C** Single correctable error.  
**AMDSMI\_RAS\_ERR\_STATE\_MULT\_UC** Multiple uncorrectable errors.  
**AMDSMI\_RAS\_ERR\_STATE\_POISON** page. Treat as uncorrectable. Firmware detected error and isolated  
**AMDSMI\_RAS\_ERR\_STATE\_ENABLED** ECC is enabled.

## 8.1.4.16 enum amdsmi\_memory\_type\_t

Types of memory.

Enumerator

**AMDSMI\_MEM\_TYPE\_VRAM** VRAM memory.  
**AMDSMI\_MEM\_TYPE\_VIS\_VRAM** VRAM memory that is visible.  
**AMDSMI\_MEM\_TYPE\_GTT** GTT memory.

## 8.1.4.17 enum amdsmi\_freq\_ind\_t

The values of this enum are used as frequency identifiers.

Enumerator

**AMDSMI\_FREQ\_IND\_MIN** Index used for the minimum frequency value.  
**AMDSMI\_FREQ\_IND\_MAX** Index used for the maximum frequency value.  
**AMDSMI\_FREQ\_IND\_INVALID** An invalid frequency index.

## 8.1.4.18 enum amdsmi\_memory\_page\_status\_t

Reserved Memory Page States.

Enumerator

**AMDSMI\_MEM\_PAGE\_STATUS\_RESERVED** and not available for use Reserved. This gpu page is reserved  
**AMDSMI\_MEM\_PAGE\_STATUS\_PENDING** Pending. This gpu page is marked as bad and will be marked reserved at the next window.  
**AMDSMI\_MEM\_PAGE\_STATUS\_UNRESERVABLE** Unable to reserve this page.

## 8.1.4.19 enum AMDSMI\_IO\_LINK\_TYPE

Types for IO Link.

Enumerator

**AMDSMI\_IOLINK\_TYPE\_UNDEFINED** unknown type.  
**AMDSMI\_IOLINK\_TYPE\_PCIEXPRESS** PCI Express.  
**AMDSMI\_IOLINK\_TYPE\_XGMI** XGMI.  
**AMDSMI\_IOLINK\_TYPE\_NUMIOLINKTYPES** Number of IO Link types.  
**AMDSMI\_IOLINK\_TYPE\_SIZE** Max of IO Link types.

## 8.1.4.20 enum AMDSMI\_UTILIZATION\_COUNTER\_TYPE

The utilization counter type.

Enumerator

**AMDSMI\_UTILIZATION\_COUNTER\_FIRST** GFX Activity.  
**AMDSMI\_COARSE\_GRAIN\_MEM\_ACTIVITY** Memory Activity.

## 8.1.5 Function Documentation

## 8.1.5.1 amdsmi\_status\_t amdsmi\_get\_device\_bdf ( amdsmi\_device\_handle device\_handle, amdsmi\_bdf\_t \* bdf )

Returns BDF of the given device.

Parameters

in	<i>device_handle</i>	Device which to query
out	<i>bdf</i>	Reference to BDF. Must be allocated by user.

Returns

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail

## 8.1.5.2 amdsmi\_status\_t amdsmi\_get\_device\_uuid ( amdsmi\_device\_handle device\_handle, unsigned int \* uuid\_length, char \* uuid )

Returns the UUID of the device.

Parameters

in	<i>device_handle</i>	Device which to query
in, out	<i>uuid_length</i>	Length of the uuid string. As input, must be equal or greater than SMI_GPU_UUID_SIZE and be allocated by user. As output it is the length of the uuid string.
out	<i>uuid</i>	Pointer to string to store the UUID. Must be allocated by user.

**Returns**

[amdsmi\\_status\\_t](#) | [AMDSMI\\_STATUS\\_SUCCESS](#) on success, non-zero on fail



# Index

AMDSMI\_CNTR\_CMD\_START  
    amdsmi.h, [121](#)

AMDSMI\_CNTR\_CMD\_STOP  
    amdsmi.h, [121](#)

AMDSMI\_COARSE\_GRAIN\_MEM\_ACTIVITY  
    amdsmi.h, [125](#)

AMDSMI\_DEFAULT\_VARIANT  
    amdsmi.h, [118](#)

AMDSMI\_DEV\_PERF\_LEVEL\_AUTO  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_DETERMINISM  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_HIGH  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_LOW  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_MANUAL  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_MCLK  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_SCLK  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_STABLE\_PEAK  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_STABLE\_STD  
    amdsmi.h, [120](#)

AMDSMI\_DEV\_PERF\_LEVEL\_UNKNOWN  
    amdsmi.h, [120](#)

AMDSMI\_EVENT\_MASK\_FROM\_INDEX  
    amdsmi.h, [118](#)

AMDSMI\_EVT\_GRP\_XGMI\_DATA\_OUT  
    amdsmi.h, [120](#)

AMDSMI\_EVT\_GRP\_XGMI  
    amdsmi.h, [120](#)

AMDSMI\_EVT\_XGMI\_0\_BEATS\_TX  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_0\_NOP\_TX  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_0\_REQUEST\_TX  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_0\_RESPONSE\_TX  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_1\_BEATS\_TX  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_1\_NOP\_TX  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_1\_REQUEST\_TX  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_1\_RESPONSE\_TX  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_DATA\_OUT\_1  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_DATA\_OUT\_2  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_DATA\_OUT\_3  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_DATA\_OUT\_4  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_XGMI\_DATA\_OUT\_5  
    amdsmi.h, [121](#)

AMDSMI\_EVT\_NOTIF\_VMFault  
    amdsmi.h, [121](#)

AMDSMI\_FREQ\_IND\_INVALID  
    amdsmi.h, [124](#)

AMDSMI\_FREQ\_IND\_MAX  
    amdsmi.h, [124](#)

AMDSMI\_FREQ\_IND\_MIN  
    amdsmi.h, [124](#)

AMDSMI\_GPU\_BLOCK\_ATHUB  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_DF  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_FUSE  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_GFX  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_HDP  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_INVALID  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_LAST  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_MMHUB  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_MP0  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_MP1  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_PCIE\_BIF  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_SDMA  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_SEM  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_SMN  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_UMC  
    amdsmi.h, [123](#)

AMDSMI\_GPU\_BLOCK\_XGMI\_WAFL  
     amdsmi.h, [123](#)  
 AMDSMI\_IO\_LINK\_TYPE  
     amdsmi.h, [124](#)  
 AMDSMI\_IOLINK\_TYPE\_NUMIOLINKTYPES  
     amdsmi.h, [125](#)  
 AMDSMI\_IOLINK\_TYPE\_PCIEXPRESS  
     amdsmi.h, [125](#)  
 AMDSMI\_IOLINK\_TYPE\_SIZE  
     amdsmi.h, [125](#)  
 AMDSMI\_IOLINK\_TYPE\_UNDEFINED  
     amdsmi.h, [125](#)  
 AMDSMI\_IOLINK\_TYPE\_XGMI  
     amdsmi.h, [125](#)  
 AMDSMI\_MAX\_DATE\_LENGTH  
     amdsmi.h, [118](#)  
 AMDSMI\_MAX\_FAN\_SPEED  
     amdsmi.h, [118](#)  
 AMDSMI\_MEM\_PAGE\_STATUS\_PENDING  
     amdsmi.h, [124](#)  
 AMDSMI\_MEM\_PAGE\_STATUS\_RESERVED  
     amdsmi.h, [124](#)  
 AMDSMI\_MEM\_PAGE\_STATUS\_UNRESERVABLE  
     amdsmi.h, [124](#)  
 AMDSMI\_MEM\_TYPE\_GTT  
     amdsmi.h, [124](#)  
 AMDSMI\_MEM\_TYPE\_VIS\_VRAM  
     amdsmi.h, [124](#)  
 AMDSMI\_MEM\_TYPE\_VRAM  
     amdsmi.h, [124](#)  
 AMDSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT  
     amdsmi.h, [123](#)  
 AMDSMI\_PWR\_PROF\_PRST\_COMPUTE\_MASK  
     amdsmi.h, [123](#)  
 AMDSMI\_PWR\_PROF\_PRST\_CUSTOM\_MASK  
     amdsmi.h, [123](#)  
 AMDSMI\_PWR\_PROF\_PRST\_LAST  
     amdsmi.h, [123](#)  
 AMDSMI\_PWR\_PROF\_PRST\_POWER\_SAVING\_MASK  
     amdsmi.h, [123](#)  
 AMDSMI\_PWR\_PROF\_PRST\_VIDEO\_MASK  
     amdsmi.h, [123](#)  
 AMDSMI\_PWR\_PROF\_PRST\_VR\_MASK  
     amdsmi.h, [123](#)  
 AMDSMI\_RAS\_ERR\_STATE\_DISABLED  
     amdsmi.h, [124](#)  
 AMDSMI\_RAS\_ERR\_STATE\_ENABLED  
     amdsmi.h, [124](#)  
 AMDSMI\_RAS\_ERR\_STATE\_MULT\_UC  
     amdsmi.h, [124](#)  
 AMDSMI\_RAS\_ERR\_STATE\_NONE  
     amdsmi.h, [124](#)  
 AMDSMI\_RAS\_ERR\_STATE\_PARITY  
     amdsmi.h, [124](#)  
 AMDSMI\_RAS\_ERR\_STATE\_POISON  
     amdsmi.h, [124](#)  
 AMDSMI\_RAS\_ERR\_STATE\_SING\_C  
     amdsmi.h, [124](#)  
 AMDSMI\_STATUS\_ADDRESS\_FAULT  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_API\_FAILED  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_BUSY  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_DRM\_ERROR  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_FAIL\_LOAD\_MODULE  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_FAIL\_LOAD\_SYMBOL  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_FILE\_ERROR  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_INIT\_ERROR  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_INPUT\_OUT\_OF\_BOUNDS  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_INSUFFICIENT\_SIZE  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_INTERNAL\_EXCEPTION  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_INTERRUPT  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_INVALID  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_IO  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_MAP\_ERROR  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_NO\_DATA  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_NO\_PERM  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_NO\_SLOT  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_NOT\_FOUND  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_NOT\_INIT  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_NOT\_SUPPORTED  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_NOT\_YET\_IMPLEMENTED  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_OUT\_OF\_RESOURCES  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_REFCOUNT\_OVERFLOW  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_RETRY  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_SUCCESS  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_TIMEOUT  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_UNEXPECTED\_DATA  
     amdsmi.h, [119](#)  
 AMDSMI\_STATUS\_UNEXPECTED\_SIZE

- amdsmi.h, [119](#)
- AMDSMI\_STATUS\_UNKNOWN\_ERROR
  - amdsmi.h, [119](#)
- AMDSMI\_SW\_COMP\_DRIVER
  - amdsmi.h, [120](#)
- AMDSMI\_TEMP\_CRIT\_MIN\_HYST
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_CRIT\_MIN
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_CRITICAL\_HYST
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_CRITICAL
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_CURRENT
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_EMERGENCY\_HYST
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_EMERGENCY
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_HIGHEST
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_LOWEST
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_MAX\_HYST
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_MAX
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_MIN\_HYST
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_MIN
  - amdsmi.h, [122](#)
- AMDSMI\_TEMP\_OFFSET
  - amdsmi.h, [122](#)
- AMDSMI\_UTILIZATION\_COUNTER\_FIRST
  - amdsmi.h, [125](#)
- AMDSMI\_UTILIZATION\_COUNTER\_TYPE
  - amdsmi.h, [125](#)
- AMDSMI\_VOLT\_AVERAGE
  - amdsmi.h, [122](#)
- AMDSMI\_VOLT\_CURRENT
  - amdsmi.h, [122](#)
- AMDSMI\_VOLT\_HIGHEST
  - amdsmi.h, [122](#)
- AMDSMI\_VOLT\_LOWEST
  - amdsmi.h, [122](#)
- AMDSMI\_VOLT\_MAX\_CRIT
  - amdsmi.h, [122](#)
- AMDSMI\_VOLT\_MAX
  - amdsmi.h, [122](#)
- AMDSMI\_VOLT\_MIN\_CRIT
  - amdsmi.h, [122](#)
- AMDSMI\_VOLT\_MIN
  - amdsmi.h, [122](#)
- AMDSMI\_VOLT\_TYPE\_INVALID
  - amdsmi.h, [123](#)
- AMDSMI\_VOLT\_TYPE\_VDDGFX
  - amdsmi.h, [123](#)
- ASIC & Board Static Information, [78](#)
- amdsmi\_get\_asic\_info, [78](#)
- amdsmi\_get\_board\_info, [78](#)
- amdsmi\_get\_caps\_info, [79](#)
- amdsmi\_get\_power\_cap\_info, [79](#)
- amdsmi\_get\_xgmi\_info, [79](#)
- amd\_metrics\_table\_header\_t, [89](#)
- amdsmi.h, [107](#)
  - AMDSMI\_CNTR\_CMD\_START, [121](#)
  - AMDSMI\_CNTR\_CMD\_STOP, [121](#)
  - AMDSMI\_COARSE\_GRAIN\_MEM\_ACTIVITY, [125](#)
  - AMDSMI\_DEFAULT\_VARIANT, [118](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_AUTO, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_DETERMINISM, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_HIGH, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_LOW, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_MANUAL, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_↔ MCLK, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_STABLE\_MIN\_S↔ CLK, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_STABLE\_PEAK, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_STABLE\_STD, [120](#)
  - AMDSMI\_DEV\_PERF\_LEVEL\_UNKNOWN, [120](#)
  - AMDSMI\_EVENT\_MASK\_FROM\_INDEX, [118](#)
  - AMDSMI\_EVNT\_GRP\_XGMI\_DATA\_OUT, [120](#)
  - AMDSMI\_EVNT\_GRP\_XGMI, [120](#)
  - AMDSMI\_EVNT\_XGMI\_0\_BEATS\_TX, [121](#)
  - AMDSMI\_EVNT\_XGMI\_0\_NOP\_TX, [121](#)
  - AMDSMI\_EVNT\_XGMI\_0\_REQUEST\_TX, [121](#)
  - AMDSMI\_EVNT\_XGMI\_0\_RESPONSE\_TX, [121](#)
  - AMDSMI\_EVNT\_XGMI\_1\_BEATS\_TX, [121](#)
  - AMDSMI\_EVNT\_XGMI\_1\_NOP\_TX, [121](#)
  - AMDSMI\_EVNT\_XGMI\_1\_REQUEST\_TX, [121](#)
  - AMDSMI\_EVNT\_XGMI\_1\_RESPONSE\_TX, [121](#)
  - AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_1, [121](#)
  - AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_2, [121](#)
  - AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_3, [121](#)
  - AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_4, [121](#)
  - AMDSMI\_EVNT\_XGMI\_DATA\_OUT\_5, [121](#)
  - AMDSMI\_EVT\_NOTIF\_VMFault, [121](#)
  - AMDSMI\_FREQ\_IND\_INVALID, [124](#)
  - AMDSMI\_FREQ\_IND\_MAX, [124](#)
  - AMDSMI\_FREQ\_IND\_MIN, [124](#)
  - AMDSMI\_GPU\_BLOCK\_ATHUB, [123](#)
  - AMDSMI\_GPU\_BLOCK\_DF, [123](#)
  - AMDSMI\_GPU\_BLOCK\_FUSE, [123](#)
  - AMDSMI\_GPU\_BLOCK\_GFX, [123](#)
  - AMDSMI\_GPU\_BLOCK\_HDP, [123](#)
  - AMDSMI\_GPU\_BLOCK\_INVALID, [123](#)
  - AMDSMI\_GPU\_BLOCK\_LAST, [123](#)
  - AMDSMI\_GPU\_BLOCK\_MMHUB, [123](#)
  - AMDSMI\_GPU\_BLOCK\_MP0, [123](#)
  - AMDSMI\_GPU\_BLOCK\_MP1, [123](#)
  - AMDSMI\_GPU\_BLOCK\_PCIE\_BIF, [123](#)

- AMDSMI\_GPU\_BLOCK\_SDMA, 123
- AMDSMI\_GPU\_BLOCK\_SEM, 123
- AMDSMI\_GPU\_BLOCK\_SMN, 123
- AMDSMI\_GPU\_BLOCK\_UMC, 123
- AMDSMI\_GPU\_BLOCK\_XGMI\_WAFL, 123
- AMDSMI\_IO\_LINK\_TYPE, 124
- AMDSMI\_IOLINK\_TYPE\_NUMIOLINKTYPES, 125
- AMDSMI\_IOLINK\_TYPE\_PCIEXPRESS, 125
- AMDSMI\_IOLINK\_TYPE\_SIZE, 125
- AMDSMI\_IOLINK\_TYPE\_UNDEFINED, 125
- AMDSMI\_IOLINK\_TYPE\_XGMI, 125
- AMDSMI\_MAX\_DATE\_LENGTH, 118
- AMDSMI\_MAX\_FAN\_SPEED, 118
- AMDSMI\_MEM\_PAGE\_STATUS\_PENDING, 124
- AMDSMI\_MEM\_PAGE\_STATUS\_RESERVED, 124
- AMDSMI\_MEM\_PAGE\_STATUS\_UNRESERVED, 124
- AMDSMI\_MEM\_TYPE\_GTT, 124
- AMDSMI\_MEM\_TYPE\_VIS\_VRAM, 124
- AMDSMI\_MEM\_TYPE\_VRAM, 124
- AMDSMI\_PWR\_PROF\_PRST\_BOOTUP\_DEFAULT, 123
- AMDSMI\_PWR\_PROF\_PRST\_COMPUTE\_MASK, 123
- AMDSMI\_PWR\_PROF\_PRST\_CUSTOM\_MASK, 123
- AMDSMI\_PWR\_PROF\_PRST\_LAST, 123
- AMDSMI\_PWR\_PROF\_PRST\_POWER\_SAVING\_MASK, 123
- AMDSMI\_PWR\_PROF\_PRST\_VIDEO\_MASK, 123
- AMDSMI\_PWR\_PROF\_PRST\_VR\_MASK, 123
- AMDSMI\_RAS\_ERR\_STATE\_DISABLED, 124
- AMDSMI\_RAS\_ERR\_STATE\_ENABLED, 124
- AMDSMI\_RAS\_ERR\_STATE\_MULT\_UC, 124
- AMDSMI\_RAS\_ERR\_STATE\_NONE, 124
- AMDSMI\_RAS\_ERR\_STATE\_PARITY, 124
- AMDSMI\_RAS\_ERR\_STATE\_POISON, 124
- AMDSMI\_RAS\_ERR\_STATE\_SING\_C, 124
- AMDSMI\_STATUS\_ADDRESS\_FAULT, 119
- AMDSMI\_STATUS\_API\_FAILED, 119
- AMDSMI\_STATUS\_BUSY, 119
- AMDSMI\_STATUS\_DRM\_ERROR, 119
- AMDSMI\_STATUS\_FAIL\_LOAD\_MODULE, 119
- AMDSMI\_STATUS\_FAIL\_LOAD\_SYMBOL, 119
- AMDSMI\_STATUS\_FILE\_ERROR, 119
- AMDSMI\_STATUS\_INIT\_ERROR, 119
- AMDSMI\_STATUS\_INPUT\_OUT\_OF\_BOUNDS, 119
- AMDSMI\_STATUS\_INSUFFICIENT\_SIZE, 119
- AMDSMI\_STATUS\_INTERNAL\_EXCEPTION, 119
- AMDSMI\_STATUS\_INTERRUPT, 119
- AMDSMI\_STATUS\_INVALID, 119
- AMDSMI\_STATUS\_IO, 119
- AMDSMI\_STATUS\_MAP\_ERROR, 119
- AMDSMI\_STATUS\_NO\_DATA, 119
- AMDSMI\_STATUS\_NO\_PERM, 119
- AMDSMI\_STATUS\_NO\_SLOT, 119
- AMDSMI\_STATUS\_NOT\_FOUND, 119
- AMDSMI\_STATUS\_NOT\_INIT, 119
- AMDSMI\_STATUS\_NOT\_SUPPORTED, 119
- AMDSMI\_STATUS\_NOT\_YET\_IMPLEMENTED, 119
- AMDSMI\_STATUS\_OUT\_OF\_RESOURCES, 119
- AMDSMI\_STATUS\_REFCOUNT\_OVERFLOW, 119
- AMDSMI\_STATUS\_RETRY, 119
- AMDSMI\_STATUS\_SUCCESS, 119
- AMDSMI\_STATUS\_TIMEOUT, 119
- AMDSMI\_STATUS\_UNEXPECTED\_DATA, 119
- AMDSMI\_STATUS\_UNEXPECTED\_SIZE, 119
- AMDSMI\_STATUS\_UNKNOWN\_ERROR, 119
- AMDSMI\_SW\_COMP\_DRIVER, 120
- AMDSMI\_TEMP\_CRIT\_MIN\_HYST, 122
- AMDSMI\_TEMP\_CRIT\_MIN, 122
- AMDSMI\_TEMP\_CRITICAL\_HYST, 122
- AMDSMI\_TEMP\_CRITICAL, 122
- AMDSMI\_TEMP\_CURRENT, 122
- AMDSMI\_TEMP\_EMERGENCY\_HYST, 122
- AMDSMI\_TEMP\_EMERGENCY, 122
- AMDSMI\_TEMP\_HIGHEST, 122
- AMDSMI\_TEMP\_LOWEST, 122
- AMDSMI\_TEMP\_MAX\_HYST, 122
- AMDSMI\_TEMP\_MAX, 122
- AMDSMI\_TEMP\_MIN\_HYST, 122
- AMDSMI\_TEMP\_MIN, 122
- AMDSMI\_TEMP\_OFFSET, 122
- AMDSMI\_UTILIZATION\_COUNTER\_FIRST, 125
- AMDSMI\_UTILIZATION\_COUNTER\_TYPE, 125
- AMDSMI\_VOLT\_AVERAGE, 122
- AMDSMI\_VOLT\_CURRENT, 122
- AMDSMI\_VOLT\_HIGHEST, 122
- AMDSMI\_VOLT\_LOWEST, 122
- AMDSMI\_VOLT\_MAX\_CRIT, 122
- AMDSMI\_VOLT\_MAX, 122
- AMDSMI\_VOLT\_MIN\_CRIT, 122
- AMDSMI\_VOLT\_MIN, 122
- AMDSMI\_VOLT\_TYPE\_INVALID, 123
- AMDSMI\_VOLT\_TYPE\_VDDGFX, 123
- amdsmi\_clk\_type\_t, 119
- amdsmi\_counter\_command\_t, 121
- amdsmi\_dev\_perf\_level\_t, 120
- amdsmi\_event\_group\_t, 120
- amdsmi\_event\_handle\_t, 119
- amdsmi\_event\_type\_t, 120
- amdsmi\_evt\_notification\_type\_t, 121
- amdsmi\_freq\_ind\_t, 124
- amdsmi\_get\_device\_bdf, 125
- amdsmi\_get\_device\_uuid, 125
- amdsmi\_gpu\_block\_t, 123
- amdsmi\_init\_flags\_t, 119
- amdsmi\_memory\_page\_status\_t, 124
- amdsmi\_memory\_type\_t, 124
- amdsmi\_power\_profile\_preset\_masks\_t, 123

- amdsmi\_ras\_err\_state\_t, 123
- amdsmi\_status\_t, 119
- amdsmi\_sw\_component\_t, 120
- amdsmi\_temperature\_metric\_t, 121
- amdsmi\_voltage\_metric\_t, 122
- amdsmi\_voltage\_type\_t, 122
- CLK\_TYPE\_DCEF, 120
- CLK\_TYPE\_DF, 120
- CLK\_TYPE\_SYS, 120
- amdsmi\_asic\_info\_t, 89
  - family, 90
- amdsmi\_bdf\_t, 90
- amdsmi\_board\_info\_t, 90
- amdsmi\_clk\_measure\_t, 91
- amdsmi\_clk\_type\_t
  - amdsmi.h, 119
- amdsmi\_control\_counter
  - Performance Counter Functions, 60
- amdsmi\_counter\_command\_t
  - amdsmi.h, 121
- amdsmi\_counter\_get\_available\_counters
  - Performance Counter Functions, 61
- amdsmi\_counter\_value\_t, 91
  - time\_enabled, 91
  - time\_running, 91
- amdsmi\_dev\_close\_supported\_func\_iterator
  - Supported Functions, 73
- amdsmi\_dev\_counter\_group\_supported
  - Performance Counter Functions, 59
- amdsmi\_dev\_create\_counter
  - Performance Counter Functions, 59
- amdsmi\_dev\_destroy\_counter
  - Performance Counter Functions, 60
- amdsmi\_dev\_get\_busy\_percent
  - Clock, Power and Performance Queries, 40
- amdsmi\_dev\_get\_drm\_render\_minor
  - Identifier Queries, 21
- amdsmi\_dev\_get\_ecc\_count
  - Error Queries, 54
- amdsmi\_dev\_get\_ecc\_enabled
  - Error Queries, 54
- amdsmi\_dev\_get\_ecc\_status
  - Error Queries, 55
- amdsmi\_dev\_get\_energy\_count
  - Power Queries, 26
- amdsmi\_dev\_get\_fan\_rpms
  - Physical State Queries, 34
- amdsmi\_dev\_get\_fan\_speed
  - Physical State Queries, 34
- amdsmi\_dev\_get\_fan\_speed\_max
  - Physical State Queries, 35
- amdsmi\_dev\_get\_gpu\_clk\_freq
  - Clock, Power and Performance Queries, 43
- amdsmi\_dev\_get\_gpu\_metrics\_info
  - Clock, Power and Performance Queries, 44
- amdsmi\_dev\_get\_id
  - Identifier Queries, 18
- amdsmi\_dev\_get\_memory\_busy\_percent
  - Memory Queries, 32
- amdsmi\_dev\_get\_memory\_reserved\_pages
  - Memory Queries, 32
- amdsmi\_dev\_get\_memory\_total
  - Memory Queries, 30
- amdsmi\_dev\_get\_memory\_usage
  - Memory Queries, 31
- amdsmi\_dev\_get\_od\_volt\_curve\_regions
  - Clock, Power and Performance Queries, 45
- amdsmi\_dev\_get\_od\_volt\_info
  - Clock, Power and Performance Queries, 43
- amdsmi\_dev\_get\_overdrive\_level
  - Clock, Power and Performance Queries, 42
- amdsmi\_dev\_get\_pci\_bandwidth
  - PCIe Queries, 22
- amdsmi\_dev\_get\_pci\_id
  - PCIe Queries, 22
- amdsmi\_dev\_get\_pci\_replay\_counter
  - PCIe Queries, 24
- amdsmi\_dev\_get\_pci\_throughput
  - PCIe Queries, 23
- amdsmi\_dev\_get\_perf\_level
  - Clock, Power and Performance Queries, 41
- amdsmi\_dev\_get\_power\_ave
  - Power Queries, 26
- amdsmi\_dev\_get\_power\_profile\_presets
  - Clock, Power and Performance Queries, 46
- amdsmi\_dev\_get\_subsystem\_id
  - Identifier Queries, 20
- amdsmi\_dev\_get\_subsystem\_name
  - Identifier Queries, 20
- amdsmi\_dev\_get\_temp\_metric
  - Physical State Queries, 35
- amdsmi\_dev\_get\_vendor\_name
  - Identifier Queries, 18
- amdsmi\_dev\_get\_volt\_metric
  - Physical State Queries, 36
- amdsmi\_dev\_get\_vram\_vendor
  - Identifier Queries, 19
- amdsmi\_dev\_open\_supported\_func\_iterator
  - Supported Functions, 71
- amdsmi\_dev\_open\_supported\_variant\_iterator
  - Supported Functions, 72
- amdsmi\_dev\_perf\_level\_t
  - amdsmi.h, 120
- amdsmi\_dev\_reset\_fan
  - Physical State Control, 37
- amdsmi\_dev\_reset\_gpu
  - Clock, Power and Performance Queries, 43
- amdsmi\_dev\_reset\_xgmi\_error
  - XGMI Functions, 65
- amdsmi\_dev\_set\_clk\_freq
  - Clock, Power and Performance Control, 50
- amdsmi\_dev\_set\_clk\_range
  - Clock, Power and Performance Queries, 44
- amdsmi\_dev\_set\_fan\_speed
  - Physical State Control, 37
- amdsmi\_dev\_set\_od\_clk\_info

- Clock, Power and Performance Queries, [45](#)
- `amdsmi_dev_set_od_volt_info`
  - Clock, Power and Performance Queries, [45](#)
- `amdsmi_dev_set_overdrive_level`
  - Clock, Power and Performance Control, [49](#)
- `amdsmi_dev_set_overdrive_level_v1`
  - Clock, Power and Performance Control, [50](#)
- `amdsmi_dev_set_pci_bandwidth`
  - PCIe Control, [25](#)
- `amdsmi_dev_set_perf_level`
  - Clock, Power and Performance Control, [48](#)
- `amdsmi_dev_set_perf_level_v1`
  - Clock, Power and Performance Control, [49](#)
- `amdsmi_dev_set_power_cap`
  - Power Control, [28](#)
- `amdsmi_dev_set_power_profile`
  - Power Control, [28](#)
- `amdsmi_dev_xgmi_error_status`
  - XGMI Functions, [65](#)
- `amdsmi_engine_usage_t`, [92](#)
- `amdsmi_error_count_t`, [92](#)
- `amdsmi_event_group_t`
  - `amdsmi.h`, [120](#)
- `amdsmi_event_handle_t`
  - `amdsmi.h`, [119](#)
- `amdsmi_event_type_t`
  - `amdsmi.h`, [120](#)
- `amdsmi_evt_notification_data_t`, [92](#)
- `amdsmi_evt_notification_type_t`
  - `amdsmi.h`, [121](#)
- `amdsmi_freq_ind_t`
  - `amdsmi.h`, [124](#)
- `amdsmi_freq_volt_region_t`, [93](#)
- `amdsmi_frequencies_t`, [93](#)
  - current, [94](#)
  - frequency, [94](#)
  - num\_supported, [94](#)
- `amdsmi_frequency_range_t`, [94](#)
- `amdsmi_func_id_value_t`, [94](#)
  - memory\_type, [95](#)
- `amdsmi_fw_info_t`, [95](#)
- `amdsmi_get_asic_info`
  - ASIC & Board Static Information, [78](#)
- `amdsmi_get_bad_page_info`
  - Memory Queries, [31](#)
- `amdsmi_get_board_info`
  - ASIC & Board Static Information, [78](#)
- `amdsmi_get_caps_info`
  - ASIC & Board Static Information, [79](#)
- `amdsmi_get_clock_measure`
  - GPU Monitoring, [83](#)
- `amdsmi_get_compute_process_gpus`
  - System Information Functions, [64](#)
- `amdsmi_get_compute_process_info`
  - System Information Functions, [63](#)
- `amdsmi_get_compute_process_info_by_pid`
  - System Information Functions, [63](#)
- `amdsmi_get_device_bdf`
  - `amdsmi.h`, [125](#)
- `amdsmi_get_device_handle_from_bdf`
  - Discovery Queries, [17](#)
- `amdsmi_get_device_handles`
  - Discovery Queries, [16](#)
- `amdsmi_get_device_type`
  - Discovery Queries, [16](#)
- `amdsmi_get_device_uuid`
  - `amdsmi.h`, [125](#)
- `amdsmi_get_driver_version`
  - SW Version Information, [77](#)
- `amdsmi_get_ecc_error_count`
  - ECC information, [87](#)
- `amdsmi_get_event_notification`
  - Event Notification Functions, [75](#)
- `amdsmi_get_func_iter_value`
  - Supported Functions, [73](#)
- `amdsmi_get_fw_info`
  - Firmware & VBIOS queries, [81](#)
- `amdsmi_get_gpu_activity`
  - GPU Monitoring, [82](#)
- `amdsmi_get_minmax_bandwidth`
  - Hardware Topology Functions, [68](#)
- `amdsmi_get_pcie_link_caps`
  - Clock, Power and Performance Queries, [41](#)
- `amdsmi_get_pcie_link_status`
  - Clock, Power and Performance Queries, [41](#)
- `amdsmi_get_power_cap_info`
  - ASIC & Board Static Information, [79](#)
- `amdsmi_get_power_measure`
  - GPU Monitoring, [82](#)
- `amdsmi_get_process_info`
  - Process information, [85](#)
- `amdsmi_get_process_list`
  - Process information, [85](#)
- `amdsmi_get_ras_block_features_enabled`
  - Memory Queries, [31](#)
- `amdsmi_get_socket_handles`
  - Discovery Queries, [15](#)
- `amdsmi_get_socket_info`
  - Discovery Queries, [16](#)
- `amdsmi_get_target_frequency_range`
  - Power Management, [84](#)
- `amdsmi_get_utilization_count`
  - Clock, Power and Performance Queries, [40](#)
- `amdsmi_get_vbios_info`
  - Firmware & VBIOS queries, [81](#)
- `amdsmi_get_version`
  - Version Queries, [52](#)
- `amdsmi_get_version_str`
  - Version Queries, [52](#)
- `amdsmi_get_vram_usage`
  - GPU Monitoring, [83](#)
- `amdsmi_get_xgmi_info`
  - ASIC & Board Static Information, [79](#)
- `amdsmi_gpu_block_t`
  - `amdsmi.h`, [123](#)
- `amdsmi_gpu_caps_t`, [96](#)



- amdsmi\_gpu\_metrics\_t, 96
- amdsmi\_init
  - Initialization and Shutdown, 13
- amdsmi\_init\_event\_notification
  - Event Notification Functions, 74
- amdsmi\_init\_flags\_t
  - amdsmi.h, 119
- amdsmi\_is\_P2P\_accessible
  - Hardware Topology Functions, 69
- amdsmi\_memory\_page\_status\_t
  - amdsmi.h, 124
- amdsmi\_memory\_type\_t
  - amdsmi.h, 124
- amdsmi\_next\_func\_iter
  - Supported Functions, 72
- amdsmi\_od\_vddc\_point\_t, 96
- amdsmi\_od\_volt\_curve\_t, 97
  - vc\_points, 97
- amdsmi\_od\_volt\_freq\_data\_t, 97
  - curr\_mclk\_range, 98
- amdsmi\_pcie\_bandwidth\_t, 98
  - lanes, 99
  - transfer\_rate, 99
- amdsmi\_pcie\_info\_t, 99
- amdsmi\_power\_cap\_info\_t, 99
- amdsmi\_power\_measure\_t, 100
- amdsmi\_power\_profile\_preset\_masks\_t
  - amdsmi.h, 123
- amdsmi\_power\_profile\_status\_t, 100
  - available\_profiles, 100
  - current, 100
  - num\_profiles, 100
- amdsmi\_proc\_info\_t, 101
  - container\_name, 101
  - engine\_usage, 101
  - memory\_usage, 101
- amdsmi\_process\_info\_t, 102
- amdsmi\_range\_t, 102
- amdsmi\_ras\_err\_state\_t
  - amdsmi.h, 123
- amdsmi\_read\_counter
  - Performance Counter Functions, 61
- amdsmi\_retired\_page\_record\_t, 103
- amdsmi\_set\_event\_notification\_mask
  - Event Notification Functions, 74
- amdsmi\_set\_perf\_determinism\_mode
  - Clock, Power and Performance Queries, 42
- amdsmi\_shut\_down
  - Initialization and Shutdown, 14
- amdsmi\_status\_string
  - Error Queries, 55
- amdsmi\_status\_t
  - amdsmi.h, 119
- amdsmi\_stop\_event\_notification
  - Event Notification Functions, 75
- amdsmi\_sw\_component\_t
  - amdsmi.h, 120
- amdsmi\_temperature\_metric\_t
  - amdsmi.h, 121
- amdsmi\_topo\_get\_link\_type
  - Hardware Topology Functions, 68
- amdsmi\_topo\_get\_link\_weight
  - Hardware Topology Functions, 67
- amdsmi\_topo\_get\_numa\_affinity
  - PCIe Queries, 23
- amdsmi\_topo\_get\_numa\_node\_number
  - Hardware Topology Functions, 67
- amdsmi\_utilization\_counter\_t, 103
- amdsmi\_vbios\_info\_t, 104
- amdsmi\_version\_t, 104
- amdsmi\_voltage\_metric\_t
  - amdsmi.h, 122
- amdsmi\_voltage\_type\_t
  - amdsmi.h, 122
- amdsmi\_vram\_info\_t, 105
- amdsmi\_xgmi\_info\_t, 105
- available\_profiles
  - amdsmi\_power\_profile\_status\_t, 100
- CLK\_TYPE\_DCEF
  - amdsmi.h, 120
- CLK\_TYPE\_DF
  - amdsmi.h, 120
- CLK\_TYPE\_SYS
  - amdsmi.h, 120
- Clock, Power and Performance Control, 48
  - amdsmi\_dev\_set\_clk\_freq, 50
  - amdsmi\_dev\_set\_overdrive\_level, 49
  - amdsmi\_dev\_set\_overdrive\_level\_v1, 50
  - amdsmi\_dev\_set\_perf\_level, 48
  - amdsmi\_dev\_set\_perf\_level\_v1, 49
- Clock, Power and Performance Queries, 39
  - amdsmi\_dev\_get\_busy\_percent, 40
  - amdsmi\_dev\_get\_gpu\_clk\_freq, 43
  - amdsmi\_dev\_get\_gpu\_metrics\_info, 44
  - amdsmi\_dev\_get\_od\_volt\_curve\_regions, 45
  - amdsmi\_dev\_get\_od\_volt\_info, 43
  - amdsmi\_dev\_get\_overdrive\_level, 42
  - amdsmi\_dev\_get\_perf\_level, 41
  - amdsmi\_dev\_get\_power\_profile\_presets, 46
  - amdsmi\_dev\_reset\_gpu, 43
  - amdsmi\_dev\_set\_clk\_range, 44
  - amdsmi\_dev\_set\_od\_clk\_info, 45
  - amdsmi\_dev\_set\_od\_volt\_info, 45
  - amdsmi\_get\_pcie\_link\_caps, 41
  - amdsmi\_get\_pcie\_link\_status, 41
  - amdsmi\_get\_utilization\_count, 40
  - amdsmi\_set\_perf\_determinism\_mode, 42
- container\_name
  - amdsmi\_proc\_info\_t, 101
- curr\_mclk\_range
  - amdsmi\_od\_volt\_freq\_data\_t, 98
- current
  - amdsmi\_frequencies\_t, 94
  - amdsmi\_power\_profile\_status\_t, 100
- Discovery Queries, 15

- amdsmi\_get\_device\_handle\_from\_bdf, 17
- amdsmi\_get\_device\_handles, 16
- amdsmi\_get\_device\_type, 16
- amdsmi\_get\_socket\_handles, 15
- amdsmi\_get\_socket\_info, 16
- ECC information, 87
  - amdsmi\_get\_ecc\_error\_count, 87
- engine\_usage
  - amdsmi\_proc\_info\_t, 101
- Error Queries, 54
  - amdsmi\_dev\_get\_ecc\_count, 54
  - amdsmi\_dev\_get\_ecc\_enabled, 54
  - amdsmi\_dev\_get\_ecc\_status, 55
  - amdsmi\_status\_string, 55
- Event Notification Functions, 74
  - amdsmi\_get\_event\_notification, 75
  - amdsmi\_init\_event\_notification, 74
  - amdsmi\_set\_event\_notification\_mask, 74
  - amdsmi\_stop\_event\_notification, 75
- family
  - amdsmi\_asic\_info\_t, 90
- Firmware & VBIOS queries, 81
  - amdsmi\_get\_fw\_info, 81
  - amdsmi\_get\_vbios\_info, 81
- frequency
  - amdsmi\_frequencies\_t, 94
- GPU Monitoring, 82
  - amdsmi\_get\_clock\_measure, 83
  - amdsmi\_get\_gpu\_activity, 82
  - amdsmi\_get\_power\_measure, 82
  - amdsmi\_get\_vram\_usage, 83
- Hardware Topology Functions, 67
  - amdsmi\_get\_minmax\_bandwidth, 68
  - amdsmi\_is\_P2P\_accessible, 69
  - amdsmi\_topo\_get\_link\_type, 68
  - amdsmi\_topo\_get\_link\_weight, 67
  - amdsmi\_topo\_get\_numa\_node\_number, 67
- Identifier Queries, 18
  - amdsmi\_dev\_get\_drm\_render\_minor, 21
  - amdsmi\_dev\_get\_id, 18
  - amdsmi\_dev\_get\_subsystem\_id, 20
  - amdsmi\_dev\_get\_subsystem\_name, 20
  - amdsmi\_dev\_get\_vendor\_name, 18
  - amdsmi\_dev\_get\_vram\_vendor, 19
- Initialization and Shutdown, 13
  - amdsmi\_init, 13
  - amdsmi\_shut\_down, 14
- lanes
  - amdsmi\_pcie\_bandwidth\_t, 99
- Memory Queries, 30
  - amdsmi\_dev\_get\_memory\_busy\_percent, 32
  - amdsmi\_dev\_get\_memory\_reserved\_pages, 32
  - amdsmi\_dev\_get\_memory\_total, 30
  - amdsmi\_dev\_get\_memory\_usage, 31
  - amdsmi\_get\_bad\_page\_info, 31
  - amdsmi\_get\_ras\_block\_features\_enabled, 31
- memory\_type
  - amdsmi\_func\_id\_value\_t, 95
- memory\_usage
  - amdsmi\_proc\_info\_t, 101
- num\_profiles
  - amdsmi\_power\_profile\_status\_t, 100
- num\_supported
  - amdsmi\_frequencies\_t, 94
- PCIe Control, 25
  - amdsmi\_dev\_set\_pcie\_bandwidth, 25
- PCIe Queries, 22
  - amdsmi\_dev\_get\_pcie\_bandwidth, 22
  - amdsmi\_dev\_get\_pcie\_id, 22
  - amdsmi\_dev\_get\_pcie\_replay\_counter, 24
  - amdsmi\_dev\_get\_pcie\_throughput, 23
  - amdsmi\_topo\_get\_numa\_affinity, 23
- Performance Counter Functions, 57
  - amdsmi\_control\_counter, 60
  - amdsmi\_counter\_get\_available\_counters, 61
  - amdsmi\_dev\_counter\_group\_supported, 59
  - amdsmi\_dev\_create\_counter, 59
  - amdsmi\_dev\_destroy\_counter, 60
  - amdsmi\_read\_counter, 61
- Physical State Control, 37
  - amdsmi\_dev\_reset\_fan, 37
  - amdsmi\_dev\_set\_fan\_speed, 37
- Physical State Queries, 34
  - amdsmi\_dev\_get\_fan\_rpms, 34
  - amdsmi\_dev\_get\_fan\_speed, 34
  - amdsmi\_dev\_get\_fan\_speed\_max, 35
  - amdsmi\_dev\_get\_temp\_metric, 35
  - amdsmi\_dev\_get\_volt\_metric, 36
- Power Control, 28
  - amdsmi\_dev\_set\_power\_cap, 28
  - amdsmi\_dev\_set\_power\_profile, 28
- Power Management, 84
  - amdsmi\_get\_target\_frequency\_range, 84
- Power Queries, 26
  - amdsmi\_dev\_get\_energy\_count, 26
  - amdsmi\_dev\_get\_power\_ave, 26
- Process information, 85
  - amdsmi\_get\_process\_info, 85
  - amdsmi\_get\_process\_list, 85
- SW Version Information, 77
  - amdsmi\_get\_driver\_version, 77
- Supported Functions, 70
  - amdsmi\_dev\_close\_supported\_func\_iterator, 73
  - amdsmi\_dev\_open\_supported\_func\_iterator, 71
  - amdsmi\_dev\_open\_supported\_variant\_iterator, 72
  - amdsmi\_get\_func\_iter\_value, 73
  - amdsmi\_next\_func\_iter, 72
- System Information Functions, 63
  - amdsmi\_get\_compute\_process\_gpus, 64



- [amdsmi\\_get\\_compute\\_process\\_info, 63](#)
  - [amdsmi\\_get\\_compute\\_process\\_info\\_by\\_pid, 63](#)
- time\_enabled
  - [amdsmi\\_counter\\_value\\_t, 91](#)
- time\_running
  - [amdsmi\\_counter\\_value\\_t, 91](#)
- transfer\_rate
  - [amdsmi\\_pcie\\_bandwidth\\_t, 99](#)
- vc\_points
  - [amdsmi\\_od\\_volt\\_curve\\_t, 97](#)
- Version Queries, [52](#)
  - [amdsmi\\_get\\_version, 52](#)
  - [amdsmi\\_get\\_version\\_str, 52](#)
- XGMI Functions, [65](#)
  - [amdsmi\\_dev\\_reset\\_xgmi\\_error, 65](#)
  - [amdsmi\\_dev\\_xgmi\\_error\\_status, 65](#)