
AMD SMI

Release 28.5.0

Advanced Micro Devices, Inc.

Jul 22, 2025

INSTALL

1	AMD SMI LIBRARY BUILD	3
1.1	Requirements	3
1.2	Build commands	3
1.3	Folder structure	4
1.4	Python wrapper	4
1.5	Python package	5
2	AMD SMI C library usage and examples	7
2.1	Key Steps for Using the AMD SMI C Library	7
2.2	AMD SMI C example	7
3	Python package usage	11
4	Overview - AMD SMI tool	13
4.1	AMD SMI tool build	13
4.2	Folder structure	13
4.3	Usage and basic commands	14
5	AMD SMI C API reference	17
5.1	File List	17
5.2	Globals	17
5.3	Data Structures	17
5.4	Data Fields	17
6	AMD SMI Python API reference	19
6.1	Requirements	19
6.2	Overview	19
6.3	Amdsmi usage	20
6.4	Amdsmi Exceptions	20
6.5	Amdsmi API	21
7	License	77

AMD SMI LIB is a library that enables you to manage and monitor AMD Virtualization Enabled GPUs. It is a thread safe, extensible C based library. The library exposes both C and Python API interface.

Some of the features that are exposed in the library are:

- Query static information about the GPU (ASIC, framebuffer, etc.)
- Query information about the FW on the physical function
- Query information about the virtual functions on the GPU
- Query temperature, clocks, and usage of the GPU
- To run the AMD SMI library, the Linux PF driver needs to be installed.

There are two parts to the AMD SMI library interface:

- **C interface**
 - Consists of C function declarations.
 - The client can call these to query information about the Linux host machine.
 - The C interface can be used by the clients to build applications in C/C++ by using this interface and library binary.
- **Python interface**
 - Consists of Python function declarations.
 - These directly call the C interface.
 - The client can use the Python interface to build applications in Python.

AMD SMI tool is a command line utility that utilizes AMD SMI Library APIs to monitor AMD GPUs. The tool is used to monitor AMD's GPUs status in a virtualization environment in both host OS and guest VM, as well as in a bare metal environment for both Windows and Linux Operating Systems. The tool outputs GPU/driver information in plain text, in JSON, or in CSV formats while it can also show the info in the console or save to the specified output file.

The tool can be used to:

- Query static information about the GPU (ASIC, framebuffer, etc.)
- Query information about the FW on the physical function
- Query information about the virtual functions on the GPU
- Query temperature, clocks, and usage of the GPU

For additional information on installation, usage, and API references, please refer to the sections below:

Install

- *Build from source*

How to

- *C library usage*
- *Python library usage*
- *CLI tool usage*

Reference

- *C API*
 - *Files*
 - *Globals*

- *Data structures*
- *Data fields*
- *Python API*

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Copyright © 2025 Advanced Micro Devices, Inc. All rights reserved.

AMD SMI LIBRARY BUILD

1.1 Requirements

Before building the integration and unit tests, ensure that `gtest` and `gmock` are installed on your system. These libraries are required for building and running the tests. You can install them using your system's package manager or build them from source. Additionally, ensure that the `lcov` package is installed on your system before running the `gen_coverage` command, as it is necessary for generating code coverage reports.

Minimum supported `lcov` version is 2.0.

1.2 Build commands

When running `make` inside the `gim` folder, the AMD SMI library is built as well. Here are some useful commands for building the AMD SMI library:

- Run `make` in the `smi-lib` folder to build the library.
- Run `sudo make install` in the `smi-lib` folder to install the compiled library (`libamdsmi.so`) and the header file (`amdsmi.h`) to the system library folder (`/usr/lib/`) and system include folder (`/usr/inc/`), respectively.
- Run `sudo make uninstall` in the `smi-lib` folder to remove the installed library (`libamdsmi.so`) and header file (`amdsmi.h`) from the system library folder (`/usr/lib/`) and system include folder (`/usr/inc/`), respectively.
- Run `make package` to create the AMD SMI Python package.
- Run `make test` to build and run the integration and unit tests.
- Run `make all` to build everything mentioned above.
- Run `make gen_coverage` to calculate the code coverage of the AMD SMI library.
- If any changes are made to the interface folder, regenerate the Python wrapper by running `make python_wrapper` and replace the `amdsmi_wrapper.py` file in the `py/interface` folder with the one generated in the build folder `build/amdsmi/amdsmi_wrapper/amdsmi_wrapper.py`.

1.2.1 Build Options

These options allow you to customize the build process, such as specifying the build type, enabling thread safety, enabling logging, and using the Thread Sanitizer.

- **BUILD_TYPE:**

This option specifies the type of build you want to perform. Common values are `Release` and `Debug`. Release builds are optimized for performance and do not include debugging information. Debug builds include debugging information and are not optimized, making them suitable for development and debugging. Default: `Release`

- **THREAD_SAFE:**

This option indicates whether the build should include thread safety features. When set to True, thread safety mechanisms (e.g., mutexes, locks) are enabled. When set to False, thread safety mechanisms are disabled, which might improve performance but can lead to race conditions in multi-threaded environments. Default: True

- **LOGGING:**

This option controls whether logging is enabled in the build. When set to True, logging code is included, which can help with debugging and monitoring. When set to False, logging code is excluded, which might improve performance. Default: False

- **THREAD_SANITIZER:**

This option indicates whether the Thread Sanitizer should be enabled. Thread Sanitizer is a tool that detects data races in multi-threaded programs. When set to True, the build includes Thread Sanitizer instrumentation. When set to False, Thread Sanitizer is not included. Default: False

- **ADDRESS_SANITIZER:**

This option indicates whether the Address Sanitizer should be enabled. Address Sanitizer is a tool that detects memory errors such as buffer overflows, use-after-free, and memory leaks. When set to True, the build includes Address Sanitizer instrumentation. When set to False, Address Sanitizer is not included. Default: False

1.3 Folder structure

Library folder structure is shown below:

```
smi-lib/
├── build/                # Contains all generated files during build
├── cli/
│   └── cpp/              # CLI C++ source code
├── dl/                  # Contains downloaded packages during build
├── drv/
│   ├── core/            # Core driver SMI code
│   ├── inc/             # Driver SMI headers
│   └── linux/           # Linux platform-specific driver code
├── examples/            # Examples of using C APIs
├── inc/                 # Internal include files
│   ├── common/          # Interface between driver and SMI library
│   └── linux/           # Header files specific to the Linux platform
├── interface/           # C API interface for clients
├── py/
│   └── interface/        # Python interface and wrapper around C APIs
├── src/                 # C implementation
├── tests/               # Library gtest/gmock tests
│   ├── integration/     # Integration tests
│   └── unit/            # Unit tests
├── utils/
│   └── scripts/         # Utility scripts
```

1.4 Python wrapper

AMD SMI stack contains a wrapper around C SMI Library. The Python API is a one-to-one mapping to the C interface of SMI Library. It exposes library functionality into Python language, allowing for fast and easy scripting.

1.4.1 Code

The Python Wrapper source code can be found in `smi-lib/py/interface` folder.

1.4.2 Build

The wrapper is built together with the SMI Library. For detailed instructions, refer to the [AMD SMI LIBRARY BUILD](#) section.

1.4.3 Code style

The code style follows the Python PEP-8 standard. See [PEP-8](#) for details.

1.4.4 Dependencies

- Requires Python 3.10 or higher.
- Python Wrapper relies on `ctypes` extension of Python language. See [CTYPES]<https://docs.python.org/3/library/ctypes.html> for details. By using this extension python code can load shared library and call its functions. The extension comes by default with Python 3.

1.4.5 Flow

1. The flow starts with the C library `libamdsmi.so`, which contains the functionality to be used in Python.
2. A Python wrapper (`amdsmi_wrapper.py`) loads the `libamdsmi.so` library and provides a Python interface to its functions.
3. Another wrapper (`amdsmi_interface.py`) provides a user-friendly interface for interacting with the library.

1.4.6 Regenerate wrapper

If any changes are made to the C interface (`smi-lib/interface/amdsmi.h`), regenerate the Python wrapper by running `make python_wrapper` and replace the `amdsmi_wrapper.py` file in the `py/interface` folder with the one generated in the build folder `build/amdsmi/amdsmi_wrapper/amdsmi_wrapper.py`.

1.5 Python package

The AMD SMI Python package is a Python package that provides a wrapper around the AMD System Management Interface (SMI) library. It allows you to interact with AMD GPUs (Graphics Processing Units) and retrieve various information and metrics related to GPU performance, temperature, power consumption, and more. To use the AMD SMI Python package, you need to have the AMD GPU driver installed on your system. The package interacts with the AMD SMI C library, to communicate with the GPU hardware. On a Linux platform, go to the `smi-lib` directory `gim/smi-lib/` and run the following commands:

```
make clean - cleaning build/ directory
make package -j$(nproc) - building AMD SMI library and getting AMD SMI Python package on the following path: gim/smi-lib/build/amdsmi/package/Release/amdsmi
```


AMD SMI C LIBRARY USAGE AND EXAMPLES

This section is dedicated to explaining how to use the AMD System Management Interface (SMI) C library. It provides guidance on the basic setup and teardown of the library, which is essential for interacting with AMD hardware through the SMI API.

The AMD SMI C library allows developers to query and control various aspects of AMD hardware, such as monitoring power usage, temperature, and performance metrics. To effectively use the library, it is important to follow the correct initialization and cleanup procedures.

2.1 Key Steps for Using the AMD SMI C Library

1. **Initialization:** Before making any calls to the AMD SMI API, the library must be initialized using the `amdsmi_init()` function. This function sets up the necessary internal data structures and prepares the library for use.
2. **Performing Operations:** Once the library is initialized, you can use the various API functions provided by AMD SMI to interact with the hardware. These functions allow you to retrieve system information, monitor hardware metrics, and perform other management tasks.
3. **Cleanup:** After completing all operations, it is crucial to call `amdsmi_shut_down()` to properly release resources and close the connection to the driver.

By following these steps, you can ensure that your application interacts with AMD hardware efficiently and safely.

For a detailed example, refer to the code snippet in the next section, which demonstrates the basic structure of an application using the AMD SMI C library.

2.2 AMD SMI C example

An application using AMD SMI must call `amdsmi_init()` to initialize the AMD SMI library before all other calls. This call initializes the internal data structures required for subsequent AMD SMI operations. In the call, a flag can be passed to indicate if the application is interested in a specific device type.

`amdsmi_shut_down()` must be the last call to properly close connection to driver and make sure that any resources held by AMD SMI are released.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "amdsmi.h"

int main(void)
```

(continues on next page)

(continued from previous page)

```

{
    amdsmi_processor_handle *processors = NULL;
    uint32_t gpu_count = 0;
    amdsmi_bdf_t bdf;
    amdsmi_asic_info_t asic;
    int ret = 0;

    // Initialize the AMD SMI library
    ret = amdsmi_init(AMDSMI_INIT_ALL_PROCESSORS);
    if (ret != AMDSMI_STATUS_SUCCESS) {
        fprintf(stderr, "Failed to initialize AMD SMI library: %d\n", ret);
        return ret;
    }

    // Get the number of GPU processors
    ret = amdsmi_get_processor_handles(NULL, &gpu_count, NULL);
    if (ret != AMDSMI_STATUS_SUCCESS) {
        fprintf(stderr, "Failed to get GPU count: %d\n", ret);
        goto fini;
    }

    // Allocate memory for processor handles
    processors = (amdsmi_processor_handle *)malloc(sizeof(amdsmi_processor_handle) * gpu_
↪count);
    if (!processors) {
        fprintf(stderr, "Memory allocation failed\n");
        ret = AMDSMI_STATUS_OUT_OF_RESOURCES;
        amdsmi_shut_down();
        return ret;
    }

    // Retrieve processor handles
    ret = amdsmi_get_processor_handles(NULL, &gpu_count, processors);
    if (ret != AMDSMI_STATUS_SUCCESS) {
        fprintf(stderr, "Failed to get processor handles: %d\n", ret);
        goto fini;
    }

    // Iterate through GPUs and retrieve ASIC information
    for (uint32_t i = 0; i < gpu_count; i++) {
        ret = amdsmi_get_gpu_device_bdf(processors[i], &bdf);
        if (ret != AMDSMI_STATUS_SUCCESS) {
            fprintf(stderr, "Failed to get GPU[%u] BDF: %d\n", i, ret);
            goto fini;
        }
        ret = amdsmi_get_gpu_asic_info(processors[i], &asic);
        if (ret != AMDSMI_STATUS_SUCCESS) {
            fprintf(stderr, "Failed to get ASIC info for GPU[%u]: %d\n", ret, i);
            goto fini;
        }
        printf("GPU[%u] BDF:[%02" PRIx64 ":%02" PRIx64 ":%02" PRIu64 "] Vendor ID: 0x%04x,
↪Device ID: 0x%04" PRIx64 " Market name: %s\n",

```

(continues on next page)

(continued from previous page)

```
        i,
        (uint64_t)bdf.bdf.bus_number, (uint64_t)bdf.bdf.device_number, (uint64_t)bdf.
↪bdf.function_number,
        asic.vendor_id,
        asic.device_id,
        asic.market_name);
    }

    fini:
    // Free allocated resources and clean up internal library resources
    free(processors);
    amdsmi_shut_down();
    return ret;
}
```


PYTHON PACKAGE USAGE

Before creating python package make sure that python version on your sistem is at least 3.10

Run `make package` to create the AMD SMI Python package. Open terminal and navigate to `gim/smi-lib/build/amdsmi/package/Release`. Run `sudo python3` command. This command will launch the Python interpreter, allowing you to execute Python code and interact with the Python environment. First, you will need to execute the following lines of code:

```
from amdsmi import * # imports all the functions and classes defined in the amdsmi_
↳package, allowing you to use them directly without specifying the package name.
amdsmi_init() # function is then called to initialize the AMD System Management_
↳Interface (SMI) library. This function sets up the necessary environment and
# resources for interacting with the AMD GPU and accessing its management information.
```

By executing these lines, you can access the functionality provided by the `amdsmi` package and start using the AMD SMI library. Most functions require you to specify the processor handle for which you want to retrieve information. In order to do that, it is necessary to call `amdsmi_get_processor_handles()` function, which will return the list of GPU device handle objects on the current machine. Executing `len(processors)` function, will be able to see the number of elements of the mentioned list:

```
processors = amdsmi_get_processor_handles()
len(processors)
```

Once you have obtained the processor handle, you can call any API that works with it. For example:

```
amdsmi_get_gpu_asic_info(processors[0])
```

Finally, call `amdsmi_shut_down()` to release the SMI library resources and close the connection to the driver:

```
amdsmi_shut_down()
```

```
from amdsmi import *

try:
    amdsmi_init()

    # amdsmi calls ...

except AmdSmiException as e:
    print(e)
finally:
    try:
```

(continues on next page)

(continued from previous page)

```
    amdsmi_shut_down()
except AmdSmiException as e:
    print(e)
```

OVERVIEW - AMD SMI TOOL

AMD SMI tool is a command line utility that utilizes AMD SMI Library APIs to monitor AMD GPUs. The tool is used to monitor AMD's GPUs status in a virtualization environment in Linux host Operating Systems and Windows guest Operating Systems. The tool outputs GPU/driver information in plain text, in JSON, or in CSV formats while it can also show the info in the console or save to the specified output file.

4.1 AMD SMI tool build

Before running the command to build the tool, make sure you are meeting the following requirements on your system:
-cmake minimum version 3.15 -g++ minimum version 8

When running make inside the gim/smi-lib/cli/cpp folder, the AMD SMI Tool will be built. Here are some useful commands for building the AMD SMI library:

- Run `make` in the gim/smi-lib/cli/cpp folder to build the tool.
- Run `make clean` to remove all files generated during the build process, such as object files and executables, to ensure clean build environment.

After build is successfully finished, navigate to gim/smi-lib/cli/cpp/build folder and tool binary should be there. Open terminal and navigate to this location and now you can execute smi tool.

4.2 Folder structure

SMI Tool folder structure is shown below:

```
cli/
├── cpp/
│   ├── cmake/                # Contains all CMake files used in the build
│   │   └── linux/
│   ├── docs/
│   │   └── external/         # Contains documents
│   ├── inc/                  # Internal include files
│   ├── src/                  # Source files
│   │   ├── guest/           # Windows Guest-specific source files
│   │   └── host/            # Host-specific source files
│   ├── test/                 # Contains integration tests
│   ├── utils/
│   │   ├── scripts/         # Utility scripts
│   │   └── third_party/
│   │       └── inc/          # Third-party libraries
```

(continues on next page)

(continued from previous page)

```
└─ json/
   tabulate/
```

4.3 Usage and basic commands

```
sudo ./amd-smi <command> <options>
```

- **<command>** is the primary command to execute. It must be the first argument after `amd-smi`.
- **<options>** can include subcommands, modifiers, or other arguments relevant to the specified command.

To get detailed information about the available commands and options, you can run `help` command. The `help` command provides a comprehensive overview of the tool's functionalities and usage instructions. Simply run tool without arguments or with command `help`.

```
$ sudo ./amd-smi help

Copyright 2023-2025 Advanced Micro Devices, Inc. All rights reserved.

usage: amd-smi help

AMD System Management Interface | AMD SMI tool version 24.16.5.0

AMD-SMI Commands:
```

	Descriptions:
version	Display version information
list	List GPU information
static	Gets static information about the specified GPU
metric	Gets metric information about the specified GPU
monitor	Monitor metrics for target devices
bad-pages	Gets bad page information about the specified GPU
event	Displays event information for the given GPU
firmware	Gets firmware information about the specified GPU
set	Set options for devices
reset	Reset options for devices
xgmi	Displays xgmi information of the devices
topology	Displays topology information of the devices
partition	Displays partition information of the devices

From help message you can see which subcommand are supported on the system and a short description for each command.

To access the help documentation for a specific command, simply use that command name followed by the `help` command. For example, if you want to get help for “list” command you can use tool the following way.

```
$ sudo ./amd-smi list --help

Copyright 2023-2024 Advanced Micro Devices, Inc. All rights reserved.

usage: amd-smi list [-h | --help] [--json | --csv] [--file FILE] [-g | --gpu [GPU ...]]

List all GPUs and VFs on the system and their most basic general information.
If no GPU is specified, returns basic information for all GPUs on the system.
```

(continues on next page)

(continued from previous page)

List arguments:

	Description:
-h, --help	show this help message and exit
-g, --gpu [GPU ...]	Select a GPU ID, BDF or UUID, if not selected it will return ↪ for all GPUs

Command Modifiers:

	Description:
--json	Displays output in JSON format (human readable by default).
--csv	Displays output in CSV format (human readable by default).
--file FILE	Saves output into a file on the provided path (stdout by default).

Here is the example of how you can now use the tool to get list command output in human(plain), json and csv format, by using modifiers.

```
$ sudo ./amd-smi list
```

```
GPU: 0
  BDF: 0000:0c:00.0
  UUID: 67ff74a1-0000-1000-8081-b5b9fd6edd00
  VF: 0
    BDF: 0000:0c:02.0
    UUID: 670074a1-0000-1000-8081-b5b9fd6edd00
```

```
$ sudo ./amd-smi list --json
```

```
[
  {
    "gpu": 0,
    "bdf": "0000:0c:00.0",
    "uuid": "67ff74a1-0000-1000-8081-b5b9fd6edd00",
    "vfs": [
      {
        "vf": 0,
        "bdf": "0000:0c:02.0",
        "uuid": "670074a1-0000-1000-8081-b5b9fd6edd00"
      }
    ]
  }
]
```

```
$ sudo ./amd-smi list --csv
```

```
gpu,gpu_bdf,gpu_uuid,vf,vf_bdf,vf_uuid
0,0000:0c:00.0,67ff74a1-0000-1000-8081-b5b9fd6edd00,0,0000:0c:02.0,670074a1-0000-1000-
↪8081-b5b9fd6edd00
```

And also all outputs can be saved to a file.

```
sudo ./amd-smi list --file=<file_name>
```


AMD SMI C API REFERENCE

This section provides comprehensive documentation for the AMD SMI C API. Explore these sections to understand the full scope of available functionalities and how to implement them in your applications.

- *Files*
- *Globals*
- *Data structures*

5.1 File List

5.2 Globals

5.3 Data Structures

5.4 Data Fields

5.4.1 Data Fields

5.4.2 Data Fields - Variables

AMD SMI PYTHON API REFERENCE

Python interface – Consists of Python function declarations, which directly call the C interface. The client can use the Python interface to build applications in Python.

6.1 Requirements

- python 3.10+ 64-bit

6.2 Overview

6.2.1 Folder structure

File Name	Note
<code>__init__.py</code>	Python package initialization file
<code>amdsmi_interface.py</code>	Amdsmi library python interface
<code>amdsmi_wrapper.py</code>	Python wrapper around amdsmi binary
<code>amdsmi_exception.py</code>	Amdsmi exceptions python file
<code>README.md</code>	Documentation

6.2.2 Build steps

Navigate to project's root folder and run Makefile command:

- `make package`

Build process will create a folder `build/amdsmi/package/BUILD_MODE/amdsmi`, where `BUILD_MODE` can be Release or Debug. The folder will contain the following files:

- `__init__.py`
- `amdsmi_interface.py`
- `amdsmi_wrapper.py`
- `amdsmi_exception.py`
- `libamdsmi.so`
- `README.md`

6.3 Amdsmi usage

Generated amdsmi folder should be copied and placed next to importing script. It should be imported as:

```
from amdsmi import *

try:
    amdsmi_init()

    # amdsmi calls ...

except AmdSmiException as e:
    print(e)
finally:
    try:
        amdsmi_shut_down()
    except AmdSmiException as e:
        print(e)
```

To initialize amdsmi lib, amdsmi_init() must be called before all other calls to amdsmi lib.

To close connection to driver, amdsmi_shut_down() must be the last call.

6.4 Amdsmi Exceptions

All exceptions are in amdsmi_exception.py file. Exceptions that can be thrown are:

- **AmdSmiException**: base smi exception class
- **AmdSmiLibraryException**: derives base **AmdSmiException** class and represents errors that can occur in smi-lib. When this exception is thrown, **err_code** and **err_info** are set. **err_code** is an integer that corresponds to errors that can occur in smi-lib and **err_info** is a string that explains the error that occurred. Example:

```
try:
    amdsmi_init()
    processors = amdsmi_get_processor_handles()
    num_of_GPUs = len(processors)
    if num_of_GPUs == 0:
        print("No GPUs on machine")
except AmdSmiException as e:
    print("Error code: {}".format(e.err_code))
    if e.err_code == AmdSmiRetCode.AMDSMI_STATUS_RETRY:
        print("Error info: {}".format(e.err_info))
finally:
    try:
        amdsmi_shut_down()
    except AmdSmiException as e:
        print(e)
```

- **AmdSmiRetryException**: Derives **AmdSmiLibraryException** class and signals processor is busy and call should be retried.
- **AmdSmiTimeoutException**: Derives **AmdSmiLibraryException** class and represents that call had timed out.
- **AmdSmiParameterException**: Derives base **AmdSmiException** class and represents errors related to invalid parameters passed to functions. When this exception is thrown, **err_msg** is set and it explains what is the actual

and expected type of the parameters.

- `AmdSmiBdfFormatException`: Derives base `AmdSmiException` class and represents invalid bdf format.

6.5 Amdsmi API

6.5.1 amdsmi_init

Description: Initialize smi lib and connect to driver

Input parameters: `init_flags` (*Optional parameter, if no value provided, default value is `AMDSMI_INIT_ALL_PROCESSORS` value*)

`init_flags` is `AmdSmiInitFlags` enum:

Field	Description
<code>INIT_ALL_PROCESSORS</code>	all processors
<code>INIT_AMD_CPUS</code>	amd cpus
<code>INIT_AMD_GPUS</code>	amd gpus
<code>INIT_NON_AMD_CPUS</code>	non amd cpus
<code>INIT_NON_AMD_GPUS</code>	non amd gpus
<code>INIT_AMD_APUS</code>	amd apus

Output: None

Exceptions that can be thrown by `amdsmi_init` function:

- `AmdSmiLibraryException`

Example:

```
try:
    amdsmi_init()
    # continue with amdsmi
except AmdSmiException as e:
    print("Init failed")
    print(e)
```

6.5.2 amdsmi_shut_down

Description: Finalize and close connection to driver

Input parameters: None

Output: None

Exceptions that can be thrown by `amdsmi_shut_down` function:

- `AmdSmiLibraryException`

Example:

```
try:
    amdsmi_shut_down()
except AmdSmiException as e:
    print("Fini failed")
    print(e)
```

6.5.3 amdsmi_get_processor_handles

Description: Returns list of GPU device handle objects on current machine

Input parameters: None

Output: List of GPU device handles

Exceptions that can be thrown by `amdsmi_get_processor_handles` function:

- `AmdSmiLibraryException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            print(amdsmi_get_gpu_device_uuid(processor))
except AmdSmiException as e:
    print(e)
```

6.5.4 amdsmi_get_processor_handle_from_bdf

Description: Returns processor handle from the given BDF

Input parameters: bdf string in form of either `<domain>:<bus>:<device>.<function>` or `<bus>:<device>.<function>` in hexcode format. Where:

- `<domain>` is 4 hex digits long from 0000-FFFF interval
- `<bus>` is 2 hex digits long from 00-FF interval
- `<device>` is 2 hex digits long from 00-1F interval
- `<function>` is 1 hex digit long from 0-7 interval

Output: processor handle object

Exceptions that can be thrown by `amdsmi_get_processor_handle_from_bdf` function:

- `AmdSmiLibraryException`
- `AmdSmiBdfFormatException`

Example:

```
try:
    processor = amdsmi_get_processor_handle_from_bdf("0000:23:00.0")
    print(amdsmi_get_gpu_device_uuid(processor))
except AmdSmiException as e:
    print(e)
```

6.5.5 amdsmi_get_gpu_device_bdf

Description: Returns BDF of the given device

Input parameters:

- GPU device for which to query

Output: BDF string in form of <domain>:<bus>:<device>.<function> in hexcode format. Where:

- <domain> is 4 hex digits long from 0000-FFFF interval
- <bus> is 2 hex digits long from 00-FF interval
- <device> is 2 hex digits long from 00-1F interval
- <function> is 1 hex digit long from 0-7 interval

Exceptions that can be thrown by `amdsmi_get_gpu_device_bdf` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    processor = amdsmi_get_processor_handle_from_bdf("0000:23:00.0")
    print("Processor's bdf:", amdsmi_get_gpu_device_bdf(processor))
except AmdSmiException as e:
    print(e)
```

6.5.6 amdsmi_get_index_from_processor_handle

Description: Returns the index of the given processor handle

Input parameters:

- GPU device for which to query

Output: GPU device index

Exceptions that can be thrown by `amdsmi_get_index_from_processor_handle` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            print("Processor's index:", amdsmi_get_index_from_processor_
handle(processor))
except AmdSmiException as e:
    print(e)
```

6.5.7 amdsmi_get_processor_handle_from_index

Description: Returns the processor handle from the given processor index

Input parameters:

- Function processor index to query

Output: processor handle object

Exceptions that can be thrown by `amdsmi_get_processor_handle_from_index` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    num_of_GPUs = len(processors)
    if num_of_GPUs == 0:
        print("No GPUs on machine")
    else:
        for index in range(num_of_GPUs):
            print("Processor handle:", amdsmi_get_processor_handle_from_index(index))
except AmdSmiException as e:
    print(e)
```

6.5.8 `amdsmi_get_vf_bdf`

Description: Returns BDF of the given VF

Input parameters:

- VF for which to query

Output: BDF string in form of `<domain>:<bus>:<device>.<function>` in hexcode format. Where:

- `<domain>` is 4 hex digits long from 0000-FFFF interval
- `<bus>` is 2 hex digits long from 00-FF interval
- `<device>` is 2 hex digits long from 00-1F interval
- `<function>` is 1 hex digit long from 0-7 interval

Exceptions that can be thrown by `amdsmi_get_vf_bdf` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    vf = amdsmi_get_vf_handle_from_bdf("0000:23:02.0")
    print("VF's bdf:", amdsmi_get_vf_bdf(vf))
except AmdSmiException as e:
    print(e)
```

6.5.9 `amdsmi_get_vf_handle_from_bdf`

Description: Returns processor handle (VF) from the given BDF

Input parameters: bdf string in form of either `<domain>:<bus>:<device>.<function>` or `<bus>:<device>.<function>` in hexcode format. Where:

- `<domain>` is 4 hex digits long from 0000-FFFF interval
- `<bus>` is 2 hex digits long from 00-FF interval

- <device> is 2 hex digits long from 00-1F interval
- <function> is 1 hex digit long from 0-7 interval

Output: processor handle object

Exceptions that can be thrown by `amdsmi_get_vf_handle_from_bdf` function:

- `AmdSmiLibraryException`
- `AmdSmiBdfFormatException`

Example:

```
try:
    vf = amdsmi_get_vf_handle_from_bdf("0000:23:02.0")
    print(amdsmi_get_vf_uuid(vf))
except AmdSmiException as e:
    print(e)
```

6.5.10 amdsmi_get_processor_handle_from_uuid

Description: Returns processor handle from the given UUID

Input parameters: uuid string Output: processor handle object

Exceptions that can be thrown by `amdsmi_get_processor_handle_from_uuid` function:

- `AmdSmiLibraryException`

Example:

```
try:
    processor = amdsmi_get_processor_handle_from_uuid("fcff7460-0000-1000-80e9-
    ↪b388cfe84658")
    print("Processor's UUID: ", amdsmi_get_gpu_device_uuid(processor))
except AmdSmiException as e:
    print(e)
```

6.5.11 amdsmi_get_vf_handle_from_uuid

Description: Returns the handle of a virtual function (VF) from the given UUID

Input parameters: uuid string Output: vf object

Exceptions that can be thrown by `amdsmi_get_vf_handle_from_uuid` function:

- `AmdSmiLibraryException`

Example:

```
try:
    vf = amdsmi_get_vf_handle_from_uuid("87007460-0000-1000-8059-3ae746ab9206")
    print("VF's UUID: ", amdsmi_get_vf_uuid(vf))
except AmdSmiException as e:
    print(e)
```

6.5.12 amdsmi_get_gpu_device_uuid

Description: Returns the UUID of the device

Input parameters:

- GPU device for which to query

Output: UUID string unique to the device

Exceptions that can be thrown by `amdsmi_get_gpu_device_uuid` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    processor = amdsmi_get_processor_handle_from_bdf("0000:23:00.0")
    print("Device UUID: ", amdsmi_get_gpu_device_uuid(processor))
except AmdSmiException as e:
    print(e)
```

6.5.13 amdsmi_get_vf_uuid

Description: Returns the UUID of the device

Input parameters:

- VF handle for which to query

Output: UUID string unique to the device

Exceptions that can be thrown by `amdsmi_get_vf_uuid` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    vf_id = amdsmi_get_vf_handle_from_vf_index(processor_handle, 0)
    print("VF UUID: ", amdsmi_get_vf_uuid(vf_id))
except AmdSmiException as e:
    print(e)
```

6.5.14 amdsmi_get_gpu_driver_info

Description: Returns the version string of the driver

Input parameters:

- `processor_handle` GPU device for which to query

Output:

- `driver_name` Driver name string that is handling the GPU device
- `driver_version` Driver version string that is handling the GPU device
- `driver_date` Driver date string that is handling the GPU device

Exceptions that can be thrown by `amdsmi_get_gpu_driver_info` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    processor = amdsmi_get_processor_handle_from_bdf("0000:23:00.0")
    driver_info = amdsmi_get_gpu_driver_info(processor)
    print("Driver name: ", driver_info.driver_name)
    print("Driver version: ", driver_info.driver_version)
    print("Driver date: ", driver_info.driver_date)
except AmdSmiException as e:
    print(e)
```

6.5.15 amdsmi_get_gpu_driver_model

Description: Returns driver model information

Input parameters:

- `processor_handle` GPU device for which to query

Output:

- current driver model from `AmdSmiDriverModelType` enum

'`AmdSmiDriverModelType`' enum:

Field	Description
WDDM	Windows Display Driver Model
WDM	Windows Driver Model
MCDM	Compute Driver Model

Exceptions that can be thrown by `amdsmi_get_gpu_driver_model` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    processor = amdsmi_get_processor_handle_from_bdf("0000:23:00.0")
    driver_model = amdsmi_get_gpu_driver_model(processor)
    print("Driver model: ", driver_model)
except AmdSmiException as e:
    print(e)
```

6.5.16 amdsmi_get_vf_handle_from_vf_index

Description: Returns VF id of the VF referenced by it's index (in partitioning info)

Input parameters:

- `processor` handle PF or child VF of a GPU device for which to query
- `VF's_index` Index of VF (0-31) in GPU's partitioning info

Output:

- VF id

Exceptions that can be thrown by `amdsmi_get_vf_handle_from_vf_index` function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    vf_id = amdsmi_get_vf_handle_from_vf_index(processor_handle, 0)
    print(amdsmi_get_vf_info(vf_id))
except AmdSmiException as e:
    print(e)
```

6.5.17 `amdsmi_get_gpu_total_ecc_count`

Description: Returns the number of ECC errors on the GPU device

Input parameters:

- `processor_handle` GPU device which to query

Output: Dictionary with fields

Field	Description
<code>correctable_count</code>	Count of ECC correctable errors
<code>uncorrectable_count</code>	Count of ECC uncorrectable errors
<code>deferred_count</code>	Count of ECC deferred errors

Exceptions that can be thrown by `amdsmi_get_gpu_total_ecc_count` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            ecc_errors = amdsmi_get_gpu_total_ecc_count(processor)
            print(ecc_errors['correctable_count'])
            print(ecc_errors['uncorrectable_count'])
            print(ecc_errors['deferred_count'])
except AmdSmiException as e:
    print(e)
```

6.5.18 amdsmi_get_gpu_ecc_count

Description: Returns the number of ECC errors on the GPU device for the given block

Input parameters:

- `processor_handle` GPU device which to query
- `block` The block for which error counts should be retrieved

`block` is `AmdSmiGpuBlock` enum:

Field	Description
UMC	UMC block
SDMA	SDMA block
GFX	GFX block
MMHUB	MMHUB block
ATHUB	ATHUB block
PCIE_BIF	PCIE_BIF block
HDP	HDP block
XGMI_WAFL	XGMI_WAFL block
DF	DF block
SMN	SMN block
SEM	SEM block
MP0	MP0 block
MP1	MP1 block
FUSE	FUSE block
MCA	MCA block
VCN	VCN block
JPEG	JPEG block
IH	IH block
MPIO	MPIO block

Output: Dictionary with fields

Field	Description
<code>correctable_count</code>	Count of ECC correctable errors
<code>uncorrectable_count</code>	Count of ECC uncorrectable errors
<code>deferred_count</code>	Count of ECC deferred errors

Exceptions that can be thrown by `amdsmi_get_gpu_ecc_count` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
```

(continues on next page)

(continued from previous page)

```

    for processor in processors:
        ecc_errors = amdsmi_get_gpu_ecc_count(processor, AmdSmiGpuBlock.UMC)
        print(ecc_errors['correctable_count'])
        print(ecc_errors['uncorrectable_count'])
        print(ecc_errors['deferred_count'])
except AmdSmiException as e:
    print(e)

```

6.5.19 amdsmi_get_gpu_ecc_enabled

Description: Returns ECC capabilities (disable/enable) for each GPU block.

Input parameters:

- `processor_handle` GPU device which to query

Output: Dictionary of each GPU block and its value (False if the block is not enabled, True if the block is enabled)

Each GPU block in the dictionary is from `AmdSmiGpuBlock` enum:

Field	Description
UMC	UMC block
SDMA	SDMA block
GFX	GFX block
MMHUB	MMHUB block
ATHUB	ATHUB block
PCIE_BIF	PCIE_BIF block
HDP	HDP block
XGMI_WAFL	XGMI_WAFL block
DF	DF block
SMN	SMN block
SEM	SEM block
MP0	MP0 block
MP1	MP1 block
FUSE	FUSE block
MCA	MCA block
VCN	VCN block
JPEG	JPEG block
IH	IH block
MPIO	MPIO block

Exceptions that can be thrown by `amdsmi_get_gpu_ecc_enabled` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```

try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:

```

(continues on next page)

(continued from previous page)

```

        print("No GPUs on machine")
    else:
        for processor in processors:
            ecc_status = amdsmi_get_gpu_ecc_enabled(processor, AmdSmiGpuBlock.UMC)
            print(ecc_status)
except AmdSmiException as e:
    print(e)

```

6.5.20 amdsmi_status_code_to_string

Description: Get a description of a provided AMDSMI error status

Input parameters:

- **status** The error status for which a description is desired

Output: String description of the provided error code

Exceptions that can be thrown by `amdsmi_status_code_to_string` function:

- `AmdSmiParameterException`

Example:

```

try:
    status_str = amdsmi_status_code_to_string(AmdSmiRetCode.SUCCESS)
    print(status_str)
except AmdSmiException as e:
    print(e)

```

6.5.21 amdsmi_get_gpu_ras_feature_info

Description: Returns RAS feature info

Input parameters:

- **processor_handle** GPU device which to query

Output: Dictionary with fields

Field	Description
<code>ras_eeeprom_version</code>	RAS EEPROM version
<code>supported_ecc_correction_sch</code>	ecc correction schema mask used with AmdSmiEccCorrectionSchemaSupport enum

Exceptions that can be thrown by `amdsmi_get_gpu_ras_feature_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```

try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            ras_feature = amdsmi_get_gpu_ras_feature_info(processor)
            print(ras_feature['ras_eeeprom_version'])
            print(ras_feature['supported_ecc_correction_schema'])
except AmdSmiException as e:
    print(e)

```

6.5.22 amdsmi_get_gpu_bad_page_info

Description: Returns bad page info.

Input parameters:

- `processor handle` object PF of a GPU device to query

Output: list of dictionaries with fields for each bad page

Field	Description
<code>retired_page</code>	64K/4K Driver managed location that is blocked from further use
<code>ts</code>	Marks the last time when the RAS event was observed
<code>mem_channel</code>	this value identifies the memory channel the issue has been reported on
<code>mcumc_id</code>	this value identifies the memory controller the issue has been reported on

Exceptions that can be thrown by `amdsmi_get_gpu_bad_page_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```

try:
    processor = amdsmi_get_processor_handle_from_bdf("0000:23.00.0")
    bad_page_info = amdsmi_get_gpu_bad_page_info(processor)
    if len(bad_page_info) == 0:
        print("no bad pages")
    else:
        for table_record in bad_page_info:
            print(hex(table_record['retired_page']))
            print(datetime.fromtimestamp(table_record['ts']).strftime('%Y/%m/%d:%H/%M/%S
→'))
            print(table_record['mem_channel'])
            print(table_record['mcumc_id'])
except AmdSmiException as e:
    print(e)

```

6.5.23 amdsmi_get_gpu_asic_info

Description: Returns asic information for the given GPU

Input parameters:

- `processor_handle` GPU device which to query

Output: Dictionary with fields

Field	Content
<code>market_name</code>	market name
<code>vendor_id</code>	vendor id
<code>vendor_name</code>	vendor name
<code>subvendor_id</code>	subsystem vendor id
<code>device_id</code>	unique id of a GPU
<code>rev_id</code>	revision id
<code>asic_serial</code>	asic serial
<code>oam_id</code>	xgmi physical id
<code>num_of_compute_units</code>	num of compute units (<i>Not supported yet, currently hardcoded to 0</i>)
<code>target_graphics_version</code>	target graphics version (<i>Not supported yet, currently hardcoded to 0</i>)
<code>subsystem_id</code>	subsystem device id

Exceptions that can be thrown by `amdsmi_get_gpu_asic_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            asic_info = amdsmi_get_gpu_asic_info(processor)
            print(asic_info['market_name'])
            print(asic_info['vendor_id'])
            print(asic_info['vendor_name'])
            print(asic_info['subvendor_id'])
            print(asic_info['device_id'])
            print(asic_info['subsystem_id'])
            print(asic_info['rev_id'])
            print(asic_info['asic_serial'])
            print(asic_info['oam_id'])
            print(asic_info['num_of_compute_units'])
            print(asic_info['target_graphics_version'])
except AmdSmiException as e:
    print(e)
```

6.5.24 amdsmi_get_pcie_info

Description: Returns static and metric information about PCIe link for the given GPU

Input parameters:

- `processor_handle` GPU device which to query

Output: Dictionary with fields

Field	Content
<code>pcie</code>	Subfield Description <code>max_pcie_width</code> Maximum number of PCIe lanes <code>max_pcie_speed</code> Maximum PCIe speed <code>pcie_interface_version</code> PCIe interface version <code>slot_type</code> Card form factor from <code>AmdSmiCardFormFactor</code> enum <code>max_pcie_interface_version</code> Maximum PCIe link generation
<code>pcie</code>	Subfield Description <code>pcie_speed</code> Current PCIe speed in MT/s <code>pcie_width</code> Current number of PCIe lanes <code>pcie_bandwidth</code> Current PCIe bandwidth in Mb/s <code>pcie_replay_count</code> Total number of the replays issued on the PCIe link <code>pcie_l0_to_recovery_count</code> Total number of times the PCIe link transitioned from L0 to the recovery state <code>pcie_replay_roll_over_count</code> Total number of replay rollovers issued on the PCIe link <code>pcie_nak_sent_count</code> Total number of NAKs issued on the PCIe link by the device <code>pcie_nak_received_count</code> Total number of NAKs issued on the PCIe link by the receiver <code>pcie_lc_perf_other_end_recovery_count</code> PCIe other end recovery counter

Exceptions that can be thrown by `amdsmi_get_pcie_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            pcie_info = amdsmi_get_pcie_info(processor)
            print(pcie_info['pcie_static']['max_pcie_width'])
            print(pcie_info['pcie_static']['max_pcie_speed'])
            print(pcie_info['pcie_static']['slot_type'])
            print(pcie_info['pcie_static']['max_pcie_interface_version'])
            print(pcie_info['pcie_metric']['pcie_speed'])
            print(pcie_info['pcie_metric']['pcie_width'])
            print(pcie_info['pcie_metric']['pcie_bandwidth'])
            print(pcie_info['pcie_metric']['pcie_interface_version'])
            print(pcie_info['pcie_metric']['pcie_replay_count'])
            print(pcie_info['pcie_metric']['pcie_l0_to_recovery_count'])
            print(pcie_info['pcie_metric']['pcie_replay_roll_over_count'])
            print(pcie_info['pcie_metric']['pcie_nak_sent_count'])
            print(pcie_info['pcie_metric']['pcie_nak_received_count'])
            print(pcie_info['pcie_metric']['pcie_lc_perf_other_end_recovery_count'])

except AmdSmiException as e:
    print(e)
```

6.5.25 amdsmi_get_power_cap_info

Description: Returns dictionary of power capabilities as currently configured on the given GPU

Input parameters:

- `processor_handle` GPU device which to query
- `sensor_ind` sensor index. Normally, this will be 0. If a processor has more than one sensor, it could be greater than 0. Parameter `sensor_ind` is unused on `@platform{host}`. It is an optional parameter and is set to 0 by default.

Output: Dictionary with fields

Field	Description
<code>power_cap</code>	power capability
<code>default_power_cap</code>	default power capability
<code>dpm_cap</code>	dynamic power management capability
<code>min_power_cap</code>	minimum power capability
<code>max_power_cap</code>	maximum power capability

Exceptions that can be thrown by `amdsmi_get_power_cap_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            power_info = amdsmi_get_power_cap_info(processor)
            print(power_info['power_cap'])
            print(power_info['default_power_cap'])
            print(power_info['dpm_cap'])
            print(power_info['min_power_cap'])
            print(power_info['max_power_cap'])
except AmdSmiException as e:
    print(e)
```

6.5.26 amdsmi_get_fb_layout

Description: Returns framebuffer related information for the given GPU

Input parameters:

- `processor_handle` PF of a GPU device for which to query

Output: Dictionary with field

Field	Description
total_fb_size	total framebuffer size in MB
pf_fb_reserved	framebuffer reserved space in MB
pf_fb_offset	framebuffer offset in MB
fb_alignment	framebuffer alignment in MB
max_vf_fb_usable	maximum usable framebuffer size in MB
min_vf_fb_usable	minimum usable framebuffer size in MB

Exceptions that can be thrown by `amdsmi_get_fb_layout` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            fb_info = amdsmi_get_fb_layout(processor)
            print(fb_info['total_fb_size'])
            print(fb_info['pf_fb_reserved'])
            print(fb_info['pf_fb_offset'])
            print(fb_info['fb_alignment'])
            print(fb_info['max_vf_fb_usable'])
            print(fb_info['min_vf_fb_usable'])
except AmdSmiException as e:
    print(e)
```

6.5.27 amdsmi_get_gpu_activity

Description: Returns the engine usage for the given GPU.

Input parameters:

- `processor_handle` GPU device which to query

Output: Dictionary with fields

Field	Description
gfx_activity	graphics engine usage/activity percentage (0 - 100)
umc_activity	memory/UMC engine usage/activity percentage (0 - 100)
mm_activity	average multimedia engine usages/activities in percentage (0 - 100)

Exceptions that can be thrown by `amdsmi_get_gpu_activity` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            engine_activity = amdsmi_get_gpu_activity(processor)
            print(engine_activity['gfx_activity'])
            print(engine_activity['umc_activity'])
            print(engine_activity['mm_activity'])
except AmdSmiException as e:
    print(e)
```

6.5.28 amdsmi_get_power_info

Description: Returns the current power, power limit, and voltage for the given GPU

Input parameters:

- `processor_handle` GPU device which to query
- `sensor_ind` sensor index. Normally, this will be 0. If a processor has more than one sensor, it could be greater than 0. Parameter `sensor_ind` is unused on `@platform{host}`. It is an optional parameter and is set to 0 by default.

Output: Dictionary with fields

Field	Description
Note: <code>socket_power</code> can rarely spike above the socket power limit in some cases	
<code>socket_power</code>	socket power
<code>gfx_voltage</code>	gfx voltage
<code>soc_voltage</code>	socket voltage
<code>mem_voltage</code>	memory voltage

Exceptions that can be thrown by `amdsmi_get_power_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            power_info = amdsmi_get_power_info(processor)
            print(power_info['socket_power'])
            print(power_info['gfx_voltage'])
            print(power_info['soc_voltage'])
```

(continues on next page)

(continued from previous page)

```

        print(power_info['mem_voltage'])
except AmdSmiException as e:
    print(e)

```

6.5.29 amdsmi_set_power_cap

Description: Sets GPU power cap.

Input parameters:

- `processor` handle processor handle
- `sensor_ind` sensor index. Normally, this will be 0. If a processor has more than one sensor, it could be greater than 0. Parameter `sensor_ind` is unused on `@platform{host}`.
- `cap` value representing power cap to set. The value must be between the minimum (`min_power_cap`) and maximum (`max_power_cap`) power cap values, which can be obtained from `::amdsmi_power_cap_info_t`.

Output:

- None

Exceptions that can be thrown by `amdsmi_set_power_cap` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```

try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        sensor_ind = 0
        for processor in processors:
            power_info = amdsmi_get_power_cap_info(processor)
            power_limit = random.randint(power_info['min_power_cap'], power_info['max_
↪power_cap'])
            amdsmi_set_power_cap(processor, sensor_ind, power_limit)

except AmdSmiException as e:
    print(e)

```

6.5.30 amdsmi_is_gpu_power_management_enabled

Description: Returns is power management enabled

Input parameters:

- `processor_handle` GPU device which to query

Output: Bool true if power management enabled else false

Exceptions that can be thrown by `amdsmi_is_gpu_power_management_enabled` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`

- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            is_power_management_enabled = amdsmi_is_gpu_power_management_
enabled(processor)
            print(is_power_management_enabled)
except AmdSmiException as e:
    print(e)
```

6.5.31 amdsmi_get_temp_metric

Description: Returns the current temperature or limit temperature for the given processor

Input parameters:

- processor_handle GPU device which to query
- thermal_domain one of AmdSmiTemperatureType enum values:

Field	Description
EDGE	edge thermal domain
HOTSPOT	hotspot/junction thermal domain
VRAM	memory/vram thermal domain
PLX	plx thermal domain (<i>Not supported yet</i>)
HBM_0	HBM 0 thermal domain (<i>Not supported yet</i>)
HBM_1	HBM 1 thermal domain (<i>Not supported yet</i>)
HBM_2	HBM 2 thermal domain (<i>Not supported yet</i>)
HBM_3	HBM 3 thermal domain (<i>Not supported yet</i>)

- thermal_metric one of AmdSmiTemperatureMetric enum values:

Field	Description
CURRENT	current thermal metric
MAX	max thermal metric (<i>Not supported yet</i>)
MIN	min thermal metric (<i>Not supported yet</i>)
MAX_HYST	max hyst thermal metric (<i>Not supported yet</i>)
MIN_HYST	min hyst thermal metric (<i>Not supported yet</i>)
CRITICAL	limit thermal metric
CRITICAL_HYST	critical hyst metric (<i>Not supported yet</i>)
EMERGENCY	emergency thermal metric (<i>Not supported yet</i>)
EMERGENCY_HYST	emergency hyst thermal metric (<i>Not supported yet</i>)
CRIT_MIN	critical min thermal metric (<i>Not supported yet</i>)
CRIT_MIN_HYST	critical min hyst thermal metric (<i>Not supported yet</i>)
OFFSET	offset thermal metric (<i>Not supported yet</i>)
LOWEST	lowest thermal metric (<i>Not supported yet</i>)
HIGHEST	highest thermal metric (<i>Not supported yet</i>)
SHUTDOWN	shutdown thermal metric

Output: Temperature value

Exceptions that can be thrown by `amdsmi_get_temp_metric` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for device in processors:
            print("===== EDGE THERMAL DOMAIN =====")
            thermal_measure = amdsmi_get_temp_metric(device, AmdSmiTemperatureType.EDGE,
↪AmdSmiTemperatureMetric.CURRENT)
            print("Current temperature:")
            print(thermal_measure)
            thermal_measure = amdsmi_get_temp_metric(device, AmdSmiTemperatureType.EDGE,
↪AmdSmiTemperatureMetric.CRITICAL)
            print("Limit temperature:")
            print(thermal_measure)
            print("===== HOTSPOT THERMAL DOMAIN =====")
            thermal_measure = amdsmi_get_temp_metric(device, AmdSmiTemperatureType.
↪HOTSPOT, AmdSmiTemperatureMetric.CURRENT)
            print("Current temperature:")
            print(thermal_measure)
            thermal_measure = amdsmi_get_temp_metric(device, AmdSmiTemperatureType.
↪HOTSPOT, AmdSmiTemperatureMetric.CRITICAL)
            print("Limit temperature:")
            print(thermal_measure)
except AmdSmiException as e:
    print(e)
```

6.5.32 amdsmi_get_gpu_cache_info

Description: Returns the cache info for the given processor

Input parameters:

- `processor_handle` GPU device which to query

Output: Dictionary with fields

Field	Description
<code>cache_size</code>	Subfield of <code>cache_properties</code> array of <code>AmdSmiCacheProperty</code>
<code>cache_level</code>	level of Compute Units
<code>sharednum_cache_instance</code>	number of instance of this cache type

`AmdSmiCacheProperty` enum values

Field	Description
ENABLED	Cache enabled
DATA_CACHE	Data cache
INST_CACHE	Inst cache
CPU_CACHE	CPU cache
SIMD_CACHE	SIMD cache

Exceptions that can be thrown by `amdsmi_get_gpu_cache_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for device in processors:
            cache_info = amdsmi_get_gpu_cache_info(device)
            for cache in cache_info["cache"]:
                print(cache["cache_properties"])
                print(cache["cache_size"])
                print(cache["cache_level"])
                print(cache["max_num_cu_shared"])
                print(cache["num_cache_instance"])
except AmdSmiException as e:
    print(e)
```

6.5.33 amdsmi_get_clock_info

Description: Returns the clock measurements for the given GPU

Input parameters:

- `processor_handle` GPU device which to query
- `clock_domain` one of `AmdSmiClkType` enum values:

Field	Description
SYS	system clock domain
GFX	gfx clock domain
DF	Data Fabric clock (for ASICs running on a separate clock) domain (<i>Not supported yet</i>)
DCEF	Display Controller Engine clock domain (<i>Not supported yet</i>)
SOC	SOC clock domain (<i>Not supported yet</i>)
MEM	memory clock domain
PCIE	PCIe clock domain (<i>Not supported yet</i>)
VCLK0	first multimedia engine (VCLK0) clock domain
VCLK1	second multimedia engine (VCLK1) clock domain
DCLK0	DCLK0 clock domain
DCLK1	DCLK1 clock domain

Output: Dictionary with fields

Field	Description
clk	current clock value for the given domain
min_clk	minimum clock value for the given domain
max_clk	maximum clock value for the given domain
clk_locked	clock locked flag only supported on GFX clock domain
clk_deep_sleep	clock deep sleep mode flag

Exceptions that can be thrown by `amdsmi_get_clock_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            print("===== GFX CLOCK DOMAIN =====")
            clock_measure = amdsmi_get_clock_info(processor, AmdSmiClkType.GFX)
            print(clock_measure['clk'])
            print(clock_measure['min_clk'])
            print(clock_measure['max_clk'])
            print(clock_measure['clk_locked'])
            print(clock_measure['clk_deep_sleep'])
            print("===== MEM CLOCK DOMAIN =====")
            clock_measure = amdsmi_get_clock_info(processor, AmdSmiClkType.MEM)
            print(clock_measure['clk'])
            print(clock_measure['min_clk'])
            print(clock_measure['max_clk'])
            print(clock_measure['clk_deep_sleep'])
            print("===== SYS CLOCK DOMAIN =====")
            clock_measure = amdsmi_get_clock_info(processor, AmdSmiClkType.SYS)
            print(clock_measure['clk'])
            print(clock_measure['min_clk'])
            print(clock_measure['max_clk'])
            print(clock_measure['clk_deep_sleep'])
            print("===== VCLK0 CLOCK DOMAIN =====")
            clock_measure = amdsmi_get_clock_info(processor, AmdSmiClkType.VCLK0)
            print(clock_measure['clk'])
            print(clock_measure['min_clk'])
            print(clock_measure['max_clk'])
            print(clock_measure['clk_deep_sleep'])
            print("===== VCLK1 CLOCK DOMAIN =====")
            clock_measure = amdsmi_get_clock_info(processor, AmdSmiClkType.VCLK1)
            print(clock_measure['clk'])
            print(clock_measure['min_clk'])
```

(continues on next page)

(continued from previous page)

```

print(clock_measure['max_clk'])
print(clock_measure['clk_deep_sleep'])
print("===== DCLK0 CLOCK DOMAIN =====")
clock_measure = amdsmi_get_clock_info(processor, AmdSmiClkType.DCLK0)
print(clock_measure['clk'])
print(clock_measure['min_clk'])
print(clock_measure['max_clk'])
print(clock_measure['clk_deep_sleep'])
print("===== DCLK1 CLOCK DOMAIN =====")
clock_measure = amdsmi_get_clock_info(processor, AmdSmiClkType.DCLK1)
print(clock_measure['clk'])
print(clock_measure['min_clk'])
print(clock_measure['max_clk'])
print(clock_measure['clk_deep_sleep'])
except AmdSmiException as e:
    print(e)

```

6.5.34 amdsmi_get_gpu_vram_info

Description: Returns the static information for the VRAM info

Input parameters:

- `processor_handle` GPU device which to query

Output: Dictionary with fields

Field	Description
<code>vram_type</code>	VRAM type from <code>AmdSmiVramType</code> enum
<code>vram_vendor</code>	VRAM vendor from <code>AmdSmiVramVendor</code> enum
<code>vram_size</code>	VRAM size in MB
<code>vram_bit_width</code>	VRAM bit width

`AmdSmiVramType` enum:

Field	Description
UNKNOWN	UNKNOWN VRAM type
HBM	HBM VRAM type
HBM2	HBM2 VRAM type
HBM2E	HBM2E VRAM type
HBM3	HBM3 VRAM type
DDR2	DDR2 VRAM type
DDR3	DDR3 VRAM type
DDR4	DDR4 VRAM type
GDDR1	GDDR1 VRAM type
GDDR2	GDDR2 VRAM type
GDDR3	GDDR3 VRAM type
GDDR4	GDDR4 VRAM type
GDDR5	GDDR5 VRAM type
GDDR6	GDDR6 VRAM type
GDDR7	GDDR7 VRAM type

AmdSmiVramVendor enum:

Field	Description
SAMSUNG	SAMSUNG VRAM vendor
INFINEON	INFINEON VRAM vendor
ELPIDA	ELPIDA VRAM vendor
ETRON	ETRON VRAM vendor
NANYA	NANYA VRAM vendor
HYNIX	HYNIX VRAM vendor
MOSEL	MOSEL VRAM vendor
WINBOND	WINBOND VRAM vendor
ESMT	ESMT VRAM vendor
MICRON	MICRON VRAM vendor
UNKNOWN	UNKNOWN VRAM vendor

Exceptions that can be thrown by `amdsmi_get_gpu_vram_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            vram_info = amdsmi_get_gpu_vram_info(processor)
            print(vram_info['vram_type'])
            print(vram_info['vram_vendor'])
            print(vram_info['vram_size'])
except AmdSmiException as e:
    print(e)
```

6.5.35 amdsmi_get_gpu_vbios_info

Description: Returns the static information for the VBIOS on the GPU device.

Input parameters:

- `processor_handle` GPU device which to query

Output: Dictionary with fields

Field	Description
<code>name</code>	vbios name
<code>build_date</code>	vbios build date
<code>part_number</code>	vbios part number
<code>version</code>	vbios version string

Exceptions that can be thrown by `amdsmi_get_gpu_vbios_info` function:

- AmdSmiLibraryException
- AmdSmiRetryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            vbios_info = amdsmi_get_gpu_vbios_info(processor)
            print(vbios_info['name'])
            print(vbios_info['build_date'])
            print(vbios_info['part_number'])
            print(vbios_info['version'])
except AmdSmiException as e:
    print(e)
```

6.5.36 amdsmi_get_fw_info

Description: Returns the firmware information for the given GPU.

Input parameters:

- processor_handle GPU device which to query

Output: Dictionary with fields

Field	Description
fw_info_list	List of dictionaries that contain information about a certain firmware block

Exceptions that can be thrown by amdsmi_get_fw_info function:

- AmdSmiLibraryException
- AmdSmiRetryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            firmware_list = amdsmi_get_fw_info(processor)
            for firmware_block in firmware_list:
                print(firmware_block['fw_id'])
                print(firmware_block['fw_version'])
except AmdSmiException as e:
    print(e)
```

6.5.37 amdsmi_get_gpu_board_info

Description: Returns board related information for the given GPU

Input parameters:

- GPU device handle object

Output: Dictionary with fields

Field	Description
model_number	board model number
product_serial	board product serial number
fru_id	fru (field-replaceable unit) id
product_name	board product name
manufacturer_name	board manufacturer name

Exceptions that can be thrown by amdsmi_get_gpu_board_info function:

- AmdSmiLibraryException
- AmdSmiRetryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            board_info = amdsmi_get_gpu_board_info(processor)
            print(board_info['model_number'])
            print(board_info['product_serial'])
            print(board_info['fru_id'])
            print(board_info['manufacturer_name'])
            print(board_info['product_name'])
except AmdSmiException as e:
    print(e)
```

6.5.38 amdsmi_get_num_vf

Description: Returns number of enabled VFs and number of supported VFs for the given GPU

Input parameters:

- processor handle PF of a GPU device for which to query

Output: Dictionary with fields

Field	Description
num_vf_enabled	number of enabled VFs
num_vf_supported	number of supported VFs

Exceptions that can be thrown by amdsmi_get_num_vf function:

- AmdSmiLibraryException
- AmdSmiRetryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            num_vf = amdsmi_get_num_vf(processor)
            print(num_vf['num_vf_enabled'])
            print(num_vf['num_vf_supported'])
except AmdSmiException as e:
    print(e)
```

6.5.39 amdsmi_get_vf_partition_info

Description: Returns array of the current framebuffer partitioning structures on the given GPU

Input parameters:

- processor handle object PF of a GPU device for which to query

Output: Array of dictionary with fields

Field	Description
vf_id	VF handle
fb	Subfield Description fb_sizeframebuffer sizefb_offsetframebuffer offset

Exceptions that can be thrown by amdsmi_get_vf_partition_info function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            partitions = amdsmi_get_vf_partition_info(processor)
            print(partitions[0]['fb']['fb_size'])
            # partitions[0]['fb']['fb_size'] is frame buffer size of the first VF on the
            ↪ given GPU
            # we can access any VF from the array via its index in partitions list
except AmdSmiException as e:
    print(e)
```

6.5.40 amdsmi_set_num_vf

Description: Set number of enabled VFs for the given GPU

Input parameters:

- `processor_handle` GPU device which to query
- number of enabled VFs to be set

Output: None

Exceptions that can be thrown by `amdsmi_set_num_vf` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            amdsmi_set_num_vf(processor, 2)
except AmdSmiException as e:
    print(e)
```

6.5.41 amdsmi_clear_vf_fb

Description: Clears framebuffer of the given VF on the given GPU.

If trying to clear the framebuffer of an active function, the call will fail

Input parameters:

- VF device handle

Output: None

Exceptions that can be thrown by `amdsmi_clear_vf_fb` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for device in devices:
            partitions = amdsmi_get_vf_partition_info(device)
            amdsmi_clear_vf_fb(partitions[0]['vf_id'])
            # partitions[0]['vf_id'] is handle of the first VF on the given GPU
```

(continues on next page)

(continued from previous page)

```
except AmdSmiException as e:
    print(e)
```

6.5.42 amdsmi_get_vf_data

Description: Returns the scheduler information and guard structure for the given VF.

Input parameters:

- VF handle

Output: Dictionary with fields

Field	Description
flr_count	function level reset counter
boot_up_t	boot up time in microseconds
shutdown_	shutdown time in microseconds
reset_tim	reset time in microseconds
state	vf state
last_boot	last boot start time
last_boot	last boot end time
last_shut	last shutdown start time
last_shut	last shutdown end time
last_rese	last reset start time
last_rese	last reset end time
current_a	current session active time, reset after guest reload
current_r	current session running time, reset after guest reload
total_act	total active time, reset after host reload
total_run	total running time, reset after host reload
enabled	show if guard info is enabled for VF
guard	Subfield Descriptionstate vf guard state amount amount of monitor events after enabled interval (dictiona: interval in seconds (sliding window) in which events are counted threshold maximum number of of events that will be processed in the given sliding window interval. Additional events during the interval elements) will be ignored.active current number of events in the interval

AmdSmiGuardType enum values are keys in guard dictionary

Field	Description
FLR	function level reset status
EXCLUSIVE_MOD	exclusive access mode status
EXCLUSIVE_TIMEOUT	exclusive access time out status
ALL_INT	generic interrupt status

State is AmdSmiGuardState enum object with values

Field	Description
NORMAL	the event number is within the threshold
FULL	the event number hits the threshold
OVERFLOW	the event number is bigger than the threshold

State is AmdSmiVfState enum object with values

Field	Description
UNAVAILABLE	vf state unavailable
AVAILABLE	vf state available
ACTIVE	vf state active
SUSPENDED	vf state suspended
FULLACCESS	vf state fullaccess
DEFAULT_AVAILABLE	same as available, indicates this is a default VF

Exceptions that can be thrown by `amdsmi_get_vf_data` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            partitions = amdsmi_get_vf_partition_info(processor)
            vf_data = amdsmi_get_vf_data(partitions[0]['vf_id'])
            sched_info = vf_data['sched']
            guard_info = vf_data['guard']
            print(sched_info['boot_up_time'])
            print(sched_info['flr_count'])
            print(sched_info['state'].name)
            print(sched_info['last_boot_start'])
            print(sched_info['last_boot_end'])
            print(sched_info['last_shutdown_start'])
            print(sched_info['last_shutdown_end'])
            print(sched_info['shutdown_time'])
            print(sched_info['last_reset_start'])
            print(sched_info['last_reset_end'])
            print(sched_info['reset_time'])
            print(sched_info['current_active_time'])
            print(sched_info['current_running_time'])
            print(sched_info['total_active_time'])
            print(sched_info['total_running_time'])

            print(guard_info['enabled'])
            for guard_type in guard_info['guard']:
                print("type: {}".format(guard_type))
                print("state: {}".format(guard_info['guard'][guard_type]['state']))
                print("amount: {}".format(guard_info['guard'][guard_type]['amount']))
                print("interval: {}".format(guard_info['guard'][guard_type]['interval']))
                print("threshold: {}".format(guard_info['guard'][guard_type]['threshold']))

            print("active: {}".format(guard_info['guard'][guard_type]['active']))
            print("=====")
```

(continues on next page)

(continued from previous page)

```
except AmdSmiException as e:
    print(e)
```

6.5.43 amdsmi_get_vf_info

Description: Returns the configuration structure for a given VF

Input parameters:

- VF handle

Output: Dictionary with fields

Field	Description
fb	Subfield Description fb_offset framebuffer offset fb_size framebuffer size
gfx_timeslice	gfx timeslice in us

Exceptions that can be thrown by amdsmi_get_vf_info function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            partitions = amdsmi_get_vf_partition_info(processor)
            # partitions[0]['vf_id'] is handle of the first VF on the given GPU
            config = amdsmi_get_vf_info(partitions[0]['vf_id'])
            print("fb_offset: {}".format(config['fb']['fb_offset']))
            print("fb_size: {}".format(config['fb']['fb_size']))

            print("gfx_timeslice : {}".format(config['gfx_timeslice']))
except AmdSmiException as e:
    print(e)
```

6.5.44 amdsmi_get_guest_data

Description: Gets guest OS information of the queried VF

Input parameters:

- processor handle VF of a GPU device

Output: Dictionary with fields

Field	Description
driver_version	driver version
fb_usage	fb usage in MB

Exceptions that can be thrown by `amdsmi_get_guest_data` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            num_vf_enabled = amdsmi_get_num_vf(processor)['num_vf_enabled']
            partitions = amdsmi_get_vf_partition_info(processor)
            for i in range(0, num_vf_enabled):
                guest_data = amdsmi_get_guest_data(partitions[i]['vf_id'])
                print(guest_data)

except AmdSmiException as e:
    print(e)
```

6.5.45 amdsmi_get_fw_error_records

Description: Gets firmware error records

Input parameters:

- `processor` handle PF of a GPU device

Output: Dictionary with field `err_records`, which is list of elements

Field	Description
<code>timestamp</code>	system time in seconds
<code>vf_idx</code>	vf index
<code>fw_id</code>	firmware id
<code>status</code>	firmware load status

Exceptions that can be thrown by `amdsmi_get_fw_error_records` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            err_records = amdsmi_get_fw_error_records(processor)
            print(err_records)
```

(continues on next page)

(continued from previous page)

```
except AmdSmiException as e:
    print(e)
```

6.5.46 amdsmi_get_dfc_fw_table

Description: Gets dfc firmware table

Input parameters:

- processor handle PF of a GPU device

Output: Dictionary with field header, and data which is a list of elements

Each header is a dictionary with following fields:

Field	Description
dfc_fw_version	dfc firmware version
dfc_fw_total_entries	number of entries in the dfc table
dfc_gart_wr_guest_min	gart wr guest min
dfc_gart_wr_guest_max	gart wr guest max

Each data entry is a dictionary with following fields:

Field	Description
dfc_fw_type	dfc firmware type
verification_enabled	verification enabled
customer_ordinal	customer ordinal
white_list	white list
black_list	black list

Each white list entry is a dictionary with following fields:

Field	Description
latest	latest
oldest	oldest

Exceptions that can be thrown by amdsmi_get_dfc_fw_table function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
```

(continues on next page)

(continued from previous page)

```

        dfc_table = amdsmi_get_dfc_fw_table(processor)
        print(dfc_table)

except AmdSmiException as e:
    print(e)

```

6.5.47 amdsmi_get_vf_fw_info

Description: Returns GPU firmware related information.

Input parameters:

- `processor` handle VF of a GPU device for which to query

Output: Dictionary with field `fw_list`, which is list of elements

If microcode of certain type is not loaded, version will be 0.

Field	Description
<code>fw_info_list</code>	Subfield Description <code>fw_id</code> <code>AmdSmiFwBlock</code> enum <code>fw_version</code> fw version which is integer

Exceptions that can be thrown by `amdsmi_get_vf_fw_info` function:

- `AmdSmiLibraryException`
- `AmdSmiRetryException`
- `AmdSmiParameterException`

Example:

```

try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            partitions = amdsmi_get_vf_partition_info(processor)
            fw_info = amdsmi_get_vf_fw_info(partitions[0]['vf_id'])
            fw_num = len(fw_info['fw_list'])
            for j in range(0, fw_num):
                fw = fw_info['fw_list'][j]
                print(fw['fw_name'].name)
                print(fw['fw_version'])
except AmdSmiException as e:
    print(e)

```

6.5.48 amdsmi_get_partition_profile_info

Description: Gets partition profile info

Input parameters:

- `processor` handle PF of a GPU device

Output: Dictionary with fields, current_profile and profiles

Field	Description
current_profile_index	current profile index
profiles	list of all profiles

Where, profiles is a list containing

Field	Description
vf_count	number of vfs
profile_caps (dictionary)	Subfield Descriptiontotal total available available optimal optimal min_value minimum max_value maximum

Keys for profile_caps dictionary are in AmdSmiProfileCapabilityType enum

Field	Description
MEMORY	memory
ENCODE	encode engine
DECODE	decode engine
COMPUTE	compute engine

Exceptions that can be thrown by amdsmi_get_partition_profile_info function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            profile_info = amdsmi_get_partition_profile_info(processor)
            print(profile_info)
except AmdSmiException as e:
    print(e)
```

6.5.49 amdsmi_get_link_metrics

Description: Gets link metric information

Input parameters:

- processor handle PF of a GPU device

Output: links list of dictionaries with fields for each link

Field	Description
bdf	BDF of the given processor
bit_rate	current link speed in Gb/s
max_bandwidth	max bandwidth of the link
link_type	type of the link from AmdSmiLinkType enum
read	total data received for each link in KB
write	total data transferred for each link in KB

AmdSmiLinkType enum:

Field	Description
INTERNAL	Unknown
XGMI	XGMI link type
PCIE	PCIe link type
NOT_APPLICABLE	Link not applicable
UNKNOWN	Unknown

Exceptions that can be thrown by `amdsmi_get_link_metrics` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            link_metrics = amdsmi_get_link_metrics(processor)
            print(link_metrics)
except AmdSmiException as e:
    print(e)
```

6.5.50 amdsmi_get_link_topology

Description: Gets link topology information between two connected processors

Input parameters:

- `source processor handle` PF of a source GPU device
- `destination processor handle` PF of a destination GPU device

Output: Dictionary with fields

Field	Description
weight	link weight between two GPUs
link_status	HW status of the link
link_type	type of the link from AmdSmiLinkType enum
num_hops	number of hops between two GPUs
fb_sharing	framebuffer sharing between two GPUs

AmdSmiLinkType enum:

Field	Description
INTERNAL	Unknown
XGMI	XGMI link type
PCIE	PCIe link type
NOT_APPLICABLE	Link not applicable
UNKNOWN	Unknown

Exceptions that can be thrown by `amdsmi_get_link_topology` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for src_processor in processors:
            for dst_processor in processors:
                link_topology = amdsmi_get_link_topology(src_processor, dst_processor)
                print(link_topology)
except AmdSmiException as e:
    print(e)
```

6.5.51 amdsmi_get_link_topology_nearest

Description: Retrieve the set of GPUs that are nearest to a given device at a specific interconnectivity level.

Input parameters:

- `processor_handle` The identifier of the given device.
- `link_type` The AmdSmiLinkType level to search for nearest devices

AmdSmiLinkType enum:

Field	Description
INTERNAL	Unknown
XGMI	XGMI link type
PCIE	PCIe link type
NOT_APPLICABLE	Link not applicable
UNKNOWN	Unknown

Output: Dictionary holding the following fields.

- `processor_list` list of all nearest processor handles found

Exceptions that can be thrown by `amdsmi_get_link_topology_nearest` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            nearest_gpus = amdsmi_get_link_topology_nearest(processor, AmdSmiLinkType.
→PCIE)

            if (len(nearest_gpus['processor_list'])) == 0:
                print("No nearest GPUs found on machine")
            else:
                print("Nearest GPUs")
                for gpu in nearest_gpus['processor_list']:
                    print(amdsmi_get_gpu_device_uuid(gpu))

except AmdSmiException as e:
    print(e)
```

6.5.52 amdsmi_get_xgmi_fb_sharing_caps

Description: Gets XGMI capabilities

Input parameters:

- `processor` handle PF of a GPU device

Output: Dictionary with fields

Field	Description
<code>mode_custom_cap</code>	flag that indicates if custom mode is supported (<i>Not supported yet</i>)
<code>mode_1_cap</code>	flag that indicates if mode_1 is supported
<code>mode_2_cap</code>	flag that indicates if mode_2 is supported
<code>mode_4_cap</code>	flag that indicates if mode_4 is supported
<code>mode_8_cap</code>	flag that indicates if mode_8 is supported

Exceptions that can be thrown by `amdsmi_get_xgmi_fb_sharing_caps` function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            caps = amdsmi_get_xgmi_fb_sharing_caps(processor)
            print(caps)

except AmdSmiException as e:
    print(e)
```

6.5.53 amdsmi_get_xgmi_fb_sharing_mode_info

Description: Gets XGMI framebuffer sharing information between two GPUs

Input parameters:

- source processor handle PF of a source GPU device
- destination processor handle PF of a destination GPU device
- mode framebuffer sharing mode from AmdSmiXgmiFbSharingMode enum

AmdSmiXgmiFbSharingMode enum:

Field	Description
CUSTOM	custom framebuffer sharing mode (<i>Not supported yet</i>)
MODE_1	framebuffer sharing mode_1
MODE_2	framebuffer sharing mode_2
MODE_4	framebuffer sharing mode_4
MODE_8	framebuffer sharing mode_8

Output: Value indicating whether framebuffer sharing is enabled between two GPUs

Exceptions that can be thrown by amdsmi_get_xgmi_fb_sharing_mode_info function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for src_processor in processors:
            for dst_processor in processors:
                fb_sharing = amdsmi_get_xgmi_fb_sharing_mode_info(src_processor, dst_
```

(continues on next page)

(continued from previous page)

```

↪processor, AmdSmiXgmiFbSharingMode.MODE_4)
    print(fb_sharing)

except AmdSmiException as e:
    print(e)

```

6.5.54 amdsmi_set_xgmi_fb_sharing_mode

Description: Sets framebuffer sharing mode

Note: This API will only work if there's no guest VM running.

Input parameters:

- processor handle PF of a GPU device
- mode framebuffer sharing mode from AmdSmiXgmiFbSharingMode enum

AmdSmiXgmiFbSharingMode enum:

Field	Description
CUSTOM	custom framebuffer sharing mode (<i>Not supported yet</i>)
MODE_1	framebuffer sharing mode_1
MODE_2	framebuffer sharing mode_2
MODE_4	framebuffer sharing mode_4
MODE_8	framebuffer sharing mode_8

Output:

- None

Exceptions that can be thrown by amdsmi_set_xgmi_fb_sharing_mode function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```

try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            amdsmi_set_xgmi_fb_sharing_mode(processor, AmdSmiXgmiFbSharingMode.MODE_4)

except AmdSmiException as e:
    print(e)

```

6.5.55 amdsmi_set_xgmi_fb_sharing_mode_v2

Description: Sets framebuffer sharing mode

Note: This API will only work if there's no guest VM running. This api can be used for custom and auto setting of xgmi frame buffer sharing. In case of custom mode: - All processors in the list must be on the same NUMA node.

Otherwise, api will return error. - If any processor from the list already belongs to an existing group, the existing group will be released automatically. In case of auto mode(MODE_X): - The input parameter processor_list[0] should be valid. Only the first element of processor_list is taken into account and it can be any gpu0,gpu1,...

Input parameters:

- processor_list list of PFs of a GPU devices
- mode framebuffer sharing mode from AmdSmiXgmiFbSharingMode enum

AmdSmiXgmiFbSharingMode enum:

Field	Description
CUSTOM	custom framebuffer sharing mode
MODE_1	framebuffer sharing mode_1
MODE_2	framebuffer sharing mode_2
MODE_4	framebuffer sharing mode_4
MODE_8	framebuffer sharing mode_8

Output:

- None

Exceptions that can be thrown by amdsmi_set_xgmi_fb_sharing_mode_v2 function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    processors_custom_mode = []
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        if len(processors) > 3:
            processors_custom_mode.append(processors[0])
            processors_custom_mode.append(processors[2])
        else:
            processors_custom_mode = processors
        amdsmi_set_xgmi_fb_sharing_mode_v2(processors_custom_mode,
        ↪ AmdSmiXgmiFbSharingMode.CUSTOM)
except AmdSmiException as e:
    print(e)
```

6.5.56 amdsmi_get_gpu_metrics

Description: Gets GPU metric information

Input parameters:

- processor handle PF of a GPU device

Output: list of dictionaries with fields for each metric

Field	Description
val	value of the metric
unit	unit of the metric from <code>AmdSmiMetricUnit</code> enum
name	name of the metric from <code>AmdSmiMetricName</code> enum
category	category of the metric from <code>AmdSmiMetricCategory</code> enum
flags	list of types of the metric from <code>AmdSmiMetricType</code> enum
vf_mask	mask of all active VFs + PF that this metric applies to

`AmdSmiMetricUnit` enum:

Field	Description
COUNTER	counter
UINT	unsigned integer
BOOL	boolean
MHZ	megahertz
PERCENT	percentage
MILLIVOLT	millivolt
CELSIUS	celsius
WATT	watt
JOULE	joule
GBPS	gigabyte per second
MBITPS	megabit per second
UNKNOWN	unknown unit

`AmdSmiMetricName` enum:

Field	Description
ACC_COUNTER	accumulated counter
FW_TIMESTAMP	firmware timestamp
CLK_GFX	gfx clock
CLK_SOC	socket clock
CLK_MEM	memory clock
CLK_VCLK	vclk clock
CLK_DCLK	dclk clock
USAGE_GFX	gfx usage
USAGE_MEM	memory usage
USAGE_MM	mm usage
USAGE_VCN	vcn usage
USAGE_JPEG	jpeg usage
VOLT_GFX	gfx voltage
VOLT_SOC	socket voltage
VOLT_MEM	memory voltage
TEMP_HOTSPOT_CURR	current hotspot temperature
TEMP_HOTSPOT_LIMIT	hotspot temperature limit
TEMP_MEM_CURR	current memory temperature
TEMP_MEM_LIMIT	memory temperature limit
TEMP_VR_CURR	current vr temperature
TEMP_SHUTDOWN	shutdown temperature
POWER_CURR	current power

continues on next page

Table 6.1 – continued from previous page

Field	Description
POWER_LIMIT	power limit
ENERGY_SOCKET	socket energy
ENERGY_CCD	ccd energy
ENERGY_XCD	xcd energy
ENERGY_AID	aid energy
ENERGY_MEM	memory energy
THROTTLE_SOCKET_ACTIVE	active socket throttle
THROTTLE_VR_ACTIVE	active vr throttle
THROTTLE_MEM_ACTIVE	active memory throttle
PCIE_BANDWIDTH	pcie bandwidth
PCIE_L0_TO_RECOVERY_COUNT	pcie l0 recovery count
PCIE_REPLAY_COUNT	pcie replay count
PCIE_REPLAY_ROLLOVER_COUNT	pcie replay rollover count
PCIE_NAK_SENT_COUNT	pcie nak sent count
PCIE_NAK_RECEIVED_COUNT	pcie nak received count
CLK_GFX_MAX_LIMIT	maximum gfx clock limit
CLK_SOC_MAX_LIMIT	maximum socket clock limit
CLK_MEM_MAX_LIMIT	maximum memory clock limit
CLK_VCLK_MAX_LIMIT	maximum vclk clock limit
CLK_DCLK_MAX_LIMIT	maximum dclk clock limit
CLK_GFX_MIN_LIMIT	minimum gfx clock limit
CLK_SOC_MIN_LIMIT	minimum socket clock limit
CLK_MEM_MIN_LIMIT	minimum memory clock limit
CLK_VCLK_MIN_LIMIT	minimum vclk clock limit
CLK_DCLK_MIN_LIMIT	minimum dclk clock limit
CLK_GFX_LOCKED	gfx clock locked
CLK_GFX_DS_DISABLED	gfx deep sleep
CLK_MEM_DS_DISABLED	memory deep sleep
CLK_SOC_DS_DISABLED	socket deep sleep
CLK_VCLK_DS_DISABLED	vclk deep sleep
CLK_DCLK_DS_DISABLED	dclk deep sleep
UNKNOWN	unknown name

AmdSmiMetricCategory enum:

Field	Description
ACC_COUNTER	counter
FREQUENCY	frequency
ACTIVITY	activity
TEMPERATURE	temperature
POWER	power
ENERGY	energy
CELSIUS	celsius
THROTTLE	throttle
PCIE	pcie
UNKNOWN	unknown category

AmdSmiMetricType enum:

Field	Description
COUNTER	counter
CHIPLET	chiplet
INST	instantaneous data
ACC	accumulated data

Exceptions that can be thrown by `amdsmi_get_gpu_metrics` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            metrics = amdsmi_get_gpu_metrics(processor)
            print(metrics)
except AmdSmiException as e:
    print(e)
```

6.5.57 amdsmi_get_soc_pstate

Description: Gets the soc pstate policy for the processor

Input parameters:

- `processor` handle PF of a GPU device

Output: Dictionary with fields

Field	Description
cur	current policy index
policies	List of policies

Each policies list entry is a dictionary with following fields:

Field	Description
policy_id	policy id
policy_description	policy description

Exceptions that can be thrown by `amdsmi_get_soc_pstate` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```

try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            dpm_policy = amdsmi_get_soc_pstate(processor)
            print(dpm_policy)
except AmdSmiException as e:
    print(e)

```

6.5.58 amdsmi_set_soc_pstate

Description: Sets the soc pstate policy for the processor

Input parameters:

- `processor` handle PF of a GPU device
- `policy_id` policy id represents one of the values we get from the policies list from `amdsmi_get_soc_pstate`.

Output:

- None

Exceptions that can be thrown by `amdsmi_set_soc_pstate` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```

try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            amdsmi_set_soc_pstate(processor, 0)
except AmdSmiException as e:
    print(e)

```

6.5.59 AmdSmiEventReader class

Description: Providing methods for event monitoring

Methods:

constructor

Description: Allocates a new event reader notifier to monitor different types of issues with the GPU

Input parameters:

- `processor` handle list list of GPU device handle objects(PFs od Vfs) for which to create event reader
- `event` category list list of the different event categories that the event reader will monitor in GPU

Event category is `AmdSmiEventCategory` enum object with values

Category	Description
NOT_USED	not used category
DRIVER	driver events(allocation, failures of APIs, debug errors)
RESET	events/notifications regarding RESET executed by the GPU
SCHED	scheduling events(world switch fail ...)
VBIOS	VBIOS events(security failures, vbios corruption...)
ECC	ecc events
PP	pp events(slave not present, dpm fail ...)
IOV	events regarding the configuration of VF resources
VF	vf events(no vbios, gpu reset fail...)
FW	events related with FW loading or FW operations
GPU	gpu fatal conditions
GUARD	guard events
GPUMON	gpumon events(fb issues ...)
MMSCH	mmsch events
XGMI	xgmi events
ALL	monitor all categories

- severity of events that can be monitored

Severity is `AmdSmiEventSeverity` enum object with values

Severity	Description
HIGH	critical error
MED	significant error
LOW	trivial error
WARN	warning
INFO	info
ALL	monitor all severity levels

severity parameter is optional. If nothing is set, events with LOW severity will be monitored by default.

Output:

- created object of `AmdSmiEventReader` class

Exceptions that can be thrown by `AmdSmiEventReader` constructor function:

- `AmdSmiParameterException`
- `AmdSmiLibraryException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        event_reader = AmdSmiEventReader(processors, {AmdSmiEventCategory.RESET})
except SMIException as e:
    print(e)
```

read

Description: Reads and return one event from event reader

Input parameters:

- `timestamp` number of microseconds to wait for an event to occur. If event does not happen monitoring is finished

Output: Dictionary with fields

Field	Description
<code>fcn_id</code>	VF handle
<code>dev_id</code>	GPU device handle
<code>timestamp</code>	UTC time (in microseconds) when the error happened
<code>data</code>	data value associated with the specific event
<code>category</code>	event category
<code>subcode</code>	event subcategory
<code>level</code>	event severity
<code>date</code>	UTC date and time when the error happend
<code>message</code>	message describing the event

event category is `AmdSmiEventCategory` enum object with values

Category	Description
<code>NOT_USED</code>	not used category
<code>DRIVER</code>	driver events(allocation, failures of APIs, debug errors)
<code>RESET</code>	events/notifications regarding RESET executed by the GPU
<code>SCHED</code>	scheduling events(world switch fail ...)
<code>VBIOS</code>	VBIOS events(security failures, vbios corruption...)
<code>ECC</code>	ecc events
<code>PP</code>	pp events(slave not present, dpm fail ...)
<code>IOV</code>	events regarding the configuration of VF resources
<code>VF</code>	vf events(no vbios, gpu reset fail...)
<code>FW</code>	events related with FW loading or FW operations
<code>GPU</code>	gpu fatal conditions
<code>GUARD</code>	guard events
<code>GPUMON</code>	gpumon events(fb issues ...)
<code>MMSCH</code>	mmsch events
<code>XGMI</code>	xgmi events
<code>ALL</code>	monitor all categories

every `AmdSmiEventCategory` has it's corresponding enum subcategory, subcategories are:

Sub-category	Field
AmdS	GPU_DEVICE_LOSTGPU_NOT_SUPPORTEDGPU_RMAGPU_NOT_INITIALIZEDGPU_MMSCH_ABNORMAL_STAT
AmdS	DRIVER_SPIN_LOCK_BUSYDRIVER_ALLOC_SYSTEM_MEM_FAILDRIVER_CREATE_GFX_WORKQUEUE_FAILD
AmdS	FW_CMD_ALLOC_BUF_FAILFW_CMD_BUF_PREP_FAILFW_RING_INIT_FAILFW_FW_APPLY_SECURITY_POLIC
AmdS	RESET_GPURESET_GPU_FAILEDRESET_FLRRESET_FLR_FAILED
AmdS	IOV_NO_GPU_IOV_CAPIOV_ASIC_NO_SRIOV_SUPPORTIOV_ENABLE_SRIOV_FAILIOV_CMD_TIMEOUTIOV_CM
AmdS	SMI_EVENT_ECC_UCESMI_EVENT_ECC_CEECC_IN_PF_FBECC_IN_CRI_REGECC_IN_VF_CRIECC_REACH_THD
AmdS	PP_SET_DPM_POLICY_FAILPP_ACTIVATE_DPM_POLICY_FAILPP_I2C_SLAVE_NOT_PRESENTPP_THROTTLER_F
AmdS	SCHED_WORLD_SWITCH_FAILSCHED_DISABLE_AUTO_HW_SWITCH_FAILSCHED_ENABLE_AUTO_HW_SWITC
AmdS	VF_ATOMBIOUS_INIT_FAILVF_NO_VBIOSVF_GPU_POST_ERRORVF_ATOMBIOUS_GET_CLOCK_FAILVF_FENCE_IN
AmdS	VBIOS_INVALIDVBIOS_IMAGE_MISSINGVBIOS_CHECKSUM_ERRVBIOS_POST_FAILVBIOS_READ_FAILVBIOS_
AmdS	GUARD_RESET_FAILGUARD_EVENT_OVERFLOW
AmdS	GPUMON_INVALID_OPTIONGPUMON_INVALID_VF_INDEXGPUMON_INVALID_FB_SIZEGPUMON_NO_SUITAB
AmdS	MMSCH_IGNORED_JOBMMMSCH_UNSUPPORTED_VCN_FW
AmdS	XGMI_TOPOLOGY_UPDATE_FAILEDXGMI_TOPOLOGY_HW_INIT_UPDATEXGMI_TOPOLOGY_UPDATE_DONEX

Severity is `AmdSmiEventSeverity` enum object with values

Severity	Description
HIGH	critical error
MED	significant error
LOW	trivial error
WARN	warning
INFO	info
ALL	monitor all severity levels

Exceptions that can be thrown by read function:

- `AmdSmiParameterException`
- `AmdSmiTimeoutException`
- `AmdSmiLibraryException`

stop

Description: Any resources used by event notification for the the given device will be freed with this function. This can be used explicitly or automatically using `with` statement, like in the examples below. This should be called either manually or automatically for every created `AmdSmiEventReader` object.

Input parameters: None

Example with manual cleanup of `AmdSmiEventReader`:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
```

(continues on next page)

(continued from previous page)

```

else:
    event_reader = AmdSmiEventReader(processors, {AmdSmiEventCategory.RESET}, ↵
↵AmdSmiEventSeverity.ALL)
    while True:
        event = event_reader.read(10*1000*1000)
        gpu_bdf = amdsmi_get_gpu_device_bdf(event['dev_id'])
        vf_bdf = amdsmi_get_gpu_device_bdf(event['fcn_id'])
        print("===== Event =====")
        print("    Time           {}".format(event['timestamp']))
        print("    Category        {}".format(event['category'].name))
        print("    Subcategory      {}".format(event['subcode'].name))
        print("    Level            {}".format(event['level'].name))
        print("    Data             {}".format(event['data']))
        print("    VF BDF           {}".format(vf_bdf))
        print("    GPU BDF          {}".format(gpu_bdf))
        print("    Date             {}".format(event['date']))
        print("    Message          {}".format(event['message']))
        print("=====")
except AmdSmiTimeoutException:
    print("No more events")
finally:
    event_reader.stop()

```

Example with automatic cleanup using with statement:

```

try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        with AmdSmiEventReader(processors, {AmdSmiEventCategory.RESET}, ↵
↵AmdSmiEventSeverity.ALL) as event_reader:
            while True:
                event = event_reader.read(10*1000*1000)
                gpu_bdf = amdsmi_get_gpu_device_bdf(event['dev_id'])
                vf_bdf = amdsmi_get_gpu_device_bdf(event['fcn_id'])
                print("===== Event =====")
                print("    Time           {}".format(event['timestamp']))
                print("    Category        {}".format(event['category'].name))
                print("    Subcategory      {}".format(event['subcode'].name))
                print("    Level            {}".format(event['level'].name))
                print("    Data             {}".format(event['data']))
                print("    VF BDF           {}".format(vf_bdf))
                print("    GPU BDF          {}".format(gpu_bdf))
                print("    Date             {}".format(event['date']))
                print("    Message          {}".format(event['message']))
                print("=====")
except AmdSmiTimeoutException:
    print("No more events")

```

6.5.60 amdsmi_get_lib_version

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

Output: amdsmi build version

Exceptions that can be thrown by amdsmi_get_lib_version function:

- AmdSmiLibraryException
- AmdSmiRetryException
- AmdSmiParameterException

Example:

```
try:
    devices = amdsmi_get_processor_handles()
    if len(devices) == 0:
        print("No GPUs on machine")
    else:
        for device in devices:
            version = amdsmi_get_lib_version()
            print(version)
except AmdSmiException as e:
    print(e)
```

6.5.61 amdsmi_get_gpu_accelerator_partition_profile_config

Description: Returns gpu accelerator partition caps as currently configured in the system

Input parameters:

- processor handle PF of a GPU device

Output: Dictionary with fields

Field	Description
resource_partitions (dictionary)	Subfield Descriptionprofile_index index of a profile resource_type type of a resource from AmdSmiAcceleratorPartitionResource enum partition_resource the resources a partition can use, which may be shared num_partitions_share_resource number of partitions that share resource of that type
default_profile_index	index of the default profile
profiles (dictionary)	Subfield Descriptionprofile_type profile type from AmdSmiAcceleratorPartitionSetting enum num_partitions number of partitions in the profile memory_caps memory capabilities of the profile profile_index index of the profile resources resources in the profile

AmdSmiAcceleratorPartitionResource enum:

Field	Description
XCC	xcc resource capabilities
ENCODER	encoder resource capabilities
DECODER	decoder resource capabilities
DMA	dma resource capabilities
JPEG	jpeg resource capabilities

AmdSmiAcceleratorPartitionSetting enum:

Field	Description
INVALID	invalid compute partition
SPX	compute partition with all xccs in group (8/1)
DPX	compute partition with four xccs in group (8/2)
TPX	compute partition with two xccs in group (6/3)
QPX	compute partition with two xccs in group (8/4)
CPX	compute partition with one xcc in group (8/8)

Exceptions that can be thrown by `amdsmi_get_gpu_accelerator_partition_profile_config` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            accelerator_partition_config = amdsmi_get_gpu_accelerator_partition_profile_
↪config(processor)
            print(accelerator_partition_config)
except AmdSmiException as e:
    print(e)
```

6.5.62 amdsmi_get_gpu_accelerator_partition_profile

Description: Returns current gpu accelerator partition cap

Input parameters:

- `processor` handle PF of a GPU device

Output: Dictionary with fields

Field	Description
<code>profile</code>	Subfield Description
<code>profile_type</code>	current profile type from <code>AmdSmiAcceleratorPartitionSetting</code>
<code>num_partitions</code>	number of partitions in the profile
<code>memory_caps</code>	memory capabilities of the profile
<code>profile_index</code>	index of the profile
<code>resources</code>	resources in the profile
<code>partition_ids</code>	array of ids for current accelerator profile

AmdSmiComputePartitionResource enum:

Field	Description
XCC	xcc resource capabilities
ENCODER	encoder resource capabilities
DECODER	decoder resource capabilities
DMA	dma resource capabilities
JPEG	jpeg resource capabilities

Exceptions that can be thrown by `amdsmi_get_gpu_accelerator_partition_profile` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            accelerator_partition_profile = amdsmi_get_gpu_accelerator_partition_
            ↪profile(processor)
            print(accelerator_partition_profile)
except AmdSmiException as e:
    print(e)
```

6.5.63 amdsmi_get_gpu_memory_partition_config

Description: Returns current gpu memory partition config and mode capabilities

Input parameters:

- `processor` handle PF of a GPU device

Output: Dictionary with fields

Field	Description
<code>partition_caps</code>	memory partition capabilities
<code>mp_mode</code>	memory partition mode from <code>AmdSmiMemoryPartitionSetting</code> enum
<code>numa_range</code> (dictionary)	Subfield Description <code>memory_type</code> memory type from <code>AmdSmiVramType</code> enum <code>start</code> start of numa range <code>end</code> end of numa range

‘`AmdSmiMemoryPartitionSetting`’ enum:

Field	Description
UNKNOWN	unknown memory partition
NPS1	memory partition with 1 number per socket
NPS2	memory partition with 2 numbers per socket
NPS4	memory partition with 4 numbers per socket
NPS8	memory partition with 8 numbers per socket

Exceptions that can be thrown by `amdsmi_get_gpu_memory_partition_config` function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            memory_partition_config = amdsmi_get_gpu_memory_partition_config(processor)
            print(memory_partition_config)
except AmdSmiException as e:
    print(e)
```

6.5.64 amdsmi_set_gpu_accelerator_partition_profile

Description: Sets accelerator partition setting based on profile_index from amdsmi_get_gpu_accelerator_partition_profile_config

Input parameters:

- processor handle PF of a GPU device
- profile_index Represents index of a partition user wants to set

Output:

- None

Exceptions that can be thrown by amdsmi_set_gpu_accelerator_partition_profile function:

- AmdSmiLibraryException
- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            amdsmi_set_gpu_accelerator_partition_profile(processor, 1)
except AmdSmiException as e:
    print(e)
```

6.5.65 amdsmi_set_gpu_memory_partition_mode

Description: Sets memory partition mode

Input parameters:

- processor handle PF of a GPU device
- setting Enum from AmdSmiMemoryPartitionSetting representing memory partitioning mode to set

AmdSmiMemoryPartitionSetting enum:

Field	Description
UNKNOWN	unknown memory partition
NPS1	memory partition with 1 number per socket
NPS2	memory partition with 2 numbers per socket
NPS4	memory partition with 4 numbers per socket
NPS8	memory partition with 8 numbers per socket

Output:

- None

Exceptions that can be thrown by `amdsmi_set_gpu_memory_partition_mode` function:

- `AmdSmiLibraryException`
- `AmdSmiParameterException`

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            amdsmi_set_gpu_memory_partition_mode(processor, AmdSmiMemoryPartitionSetting.
↪NPS1)
except AmdSmiException as e:
    print(e)
```

6.5.66 amdsmi_get_gpu_cper_entries

Description: Get gpu ras cper entries

Input parameters:

- `processor` handle PF of a GPU device
- `severity_mask` Represents different severity masks from ‘`AmdSmiCperErrorSeverity`’ enum on which filerting of cpers is based.

Field	Description
NON_FATAL_UNCORRECTED	filters non-fatal-uncorrected cpers
FATAL	filters fatal cpers
NON_FATAL_CORRECTED	filters non_fatal_corrected cpers
NUM	shows all cper types

Output:

- List of all cper errors. Each list element contains binary raw data

Exceptions that can be thrown by `amdsmi_get_gpu_cper_entries` function:

- `AmdSmiLibraryException`

- AmdSmiParameterException

Example:

```
try:
    processors = amdsmi_get_processor_handles()
    if len(processors) == 0:
        print("No GPUs on machine")
    else:
        for processor in processors:
            cper_list = amdsmi_get_gpu_cper_entries(processor, AmdSmiCperErrorSeverity.
↪NUM)
except AmdSmiException as e:
    print(e)
```


LICENSE

Copyright (c) 2025 Advanced Micro Devices, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.