E-SMI Library: EPYC™ Systems Management Interface Library

Generated by Doxygen 1.8.13

Contents

Chapter 1

EPYC™ System Management Interface (E-SMI) In-band Library

The EPYC™ System Management Interface In-band Library, or E-SMI library, is part of the EPYC™ System Management Inband software stack. It is a C library for Linux that provides a user space interface to monitor and control the CPU's Power, Energy and Performance.

Important note about Versioning and Backward Compatibility

The E-SMI library is currently under development, and therefore subject to change at the API level. The intention is to keep the API as stable as possible while in development, but in some cases we may need to break backwards compatibility in order to achieve future stability and usability. Following Semantic Versioning rules, while the E-SMI library is in a high state of change, the major version will remain 0, and achieving backward compatibility may not be possible.

Once new development has leveled off, the major version will become greater than 0, and backward compatibility will be enforced between major versions.

Building E-SMI

Additional Required software for building

In order to build the E-SMI library, the following components are required. Note that the software versions listed are what is being used in development. Earlier versions are not guaranteed to work:

- CMake (v3.5.0)

Dowloading the source

The source code for E-SMI library is available on `Github`.

Directory stucture of the source

Once the E-SMI library source has been cloned to a local Linux machine, the directory structure of source is as below:

- `$ docs/` Contains Doxygen configuration files and Library descriptions
- `$ example/` Contains e-smi tool, based on the E-SMI library
- `$ include/` Contains the header files used by the E-SMI library
- `$ src/` Contains library E-SMI source

Building the library is achieved by following the typical CMake build sequence, as follows.

```
$ mkdir -p build

$ cd build

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

$ make
```

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

Building the Documentation

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

```
$ make doc

$ cd latex

$ make
```

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

Dependencies

The E-SMI Library depends on the following device drivers from Linux to manage the system management features.

Monitoring Energy counters

The Energy counters are exposed via the RAPL MSRs and the AMD Energy driver exposes the per core and per socket information via the HWMON sys entries. The AMD Energy driver is upstreamed and available as part of Linux v5.8, this driver may be insmoded as a module.

Monitoring and Managing Power metrics, Boostlimits

The power metrics and Boostlimits features are managed by the SMU firmware and exposed via SMN PCI config space. AMD provided Linux HSMP driver exposes this information to the user-space via sys entries.

Usage Basics

Device Indices

Many of the functions in the library take a "core/socket index". The core/socket index is a number greater than or equal to 0, and less than the number of cores/sockets on the system.

Hello E-SMI

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

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

Below is a simple "Hello World" type program that display the Average Power of Sockets.

```c
#include <stdio.h>
#include <stdint.h>
#include <e_smi/e_smi.h>
#include <e_smi/e_smi_monitor.h>

int main()
{
    esmi_status_t ret;
    unsigned int i;
    uint32_t power;
    uint32_t total_sockets = 0;

    ret = esmi_init();
    if (ret != ESMI_SUCCESS) {
        printf("ESMI Not initialized, drivers not found.\n"
            "Err[%d]: %s\n", ret, esmi_get_err_msg(ret));
        return ret;
    }

    total_sockets = esmi_get_number_of_sockets();
    for (i = 0; i < total_sockets; i++) {
        power = 0;
        ret = esmi_socket_power_avg_get(i, &power);
        if (ret != ESMI_SUCCESS) {
            printf("Failed to get socket[%d] avg_power, "
                "Err[%d]:%s\n", i, ret, esmi_get_err_msg(ret));
        }
        printf("socket_%d_avgpower = %.3f Watts\n",
            i, (double)power/1000);
    }
    esmi_exit();

    return ret;
}
```

Usage

Tool Usage

E-SMI tool is a C program based on the E-SMI In-band Library, the executable "e_smi_tool" will be generated in the build/ folder. This tool provides options to Monitor and Control System Management functionality.

Below is a sample usage to dump the functionality, with default core/socket/package as 0.

```
e_smi_library/build> ./e_smi_tool

===============EPYC System Management Interface===============

_TOPOLOGY       | Count |
#CPUS           |   256 |
#SOCKETS        |     2 |

Considered Default 'CORE/SOCKET/PKG ID's are 0
_SENSOR NAME            |       Value Units         |
_CORE_ENERGY            |      3156295 uJoules      |
_SOCKET_ENERGY          |  38700978759 uJoules      |
_SOCKET_AVG_POWER       |       56.220 Watts        |
_SOCKET_POWERCAP        |      200.000 Watts        |
_SOCKET_MAX_POWERCAP    |      240.000 Watts        |
_CORE_BOOSTLIMIT        |         3200 MHz          |

======================End of EPYC SMI Log=====================

Try './e_smi_tool --help' for more information.
```

For detailed and up to date usage information, we recommend consulting the help:

For convenience purposes, following is the output from the -h flag:

```
e_smi_library/build> ./e_smi_tool --help
Usage: ./e_smi_tool [Option<s>] SOURCES
Option<s>:
    -A, (--showall)                     Get all esmi parameter Values
    -e, (--showcoreenergy)    [CORENUM]         Get energy for a given CPU
    -s, (--showsocketenergy)  [SOCKETNUM]           Get energy for a given socket
    -p, (--showsocketpower)   [SOCKETNUM]           Get power params for a given socket
    -L, (--showcoreboostlimit)[CORENUM]         Get Boostlimit for a given CPU
    -C, (--setpowercap)       [SOCKETNUM] [POWERVALUE]      Set power Cap for a given socket
    -a, (--setcoreboostlimit) [CORENUM] [BOOSTVALUE]    Set boost limit for a given core
    -b, (--setsocketboostlimit)[SOCKETNUM] [BOOSTVALUE]     Set Boost limit for a given Socket
    -c, (--setpkgboostlimit)  [PKG_BOOSTLIMIT]      Set Boost limit for a given package
    -h, (--help)                        Show this help message
```

Below is a sample usage to get the individual library functionality API's. We can pass arguments either any of the ways "./e_smi_tool -e 0" or "./e_smi_tool --showcoreenergy=0"

```
1.  e_smi_library/build> ./e_smi_tool -e 0
    ===============EPYC System Management Interface===============

    hwmon/core_energy[0]_input:     505525 uJoules

    ======================End of EPYC SMI Log=====================

2.  e_smi_library/build> ./e_smi_tool --showcoreenergy=0
    ===============EPYC System Management Interface===============

    hwmon/core_energy[0]_input:     41505525 uJoules

    ======================End of EPYC SMI Log=====================

3.      e_smi_library/build> ./e_smi_tool -e 12 --showsocketpower=1 --setpowercap 1 230000 -p 1

    ===============EPYC System Management Interface===============

    hwmon/core_energy[12]_input:      651357 uJoules
    socket[1]/avg_power     :         54.218 Watts
    socket[1]/power_cap     :         220.000 Watts
    socket[1]/power_cap_max :         240.000 Watts
    Set socket[1]/power_cap :         230.000 Watts successfully
    socket[1]/avg_power     :         55.178 Watts
    socket[1]/power_cap     :         230.000 Watts
    socket[1]/power_cap_max :         240.000 Watts

    ======================End of EPYC SMI Log=====================
```

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Module Documentation

4.1 Initialization and Shutdown

Functions

- esmi_status_t esmi_init (void)

  Initialize monitor paths.
- void esmi_exit (void)

  Clean up allocation during init.

4.1.1 Detailed Description

This function initializes the monitor paths to be used by the APIs.

4.1.2 Function Documentation

4.1.2.1 esmi_init()

```
esmi_status_t esmi_init (
             void  )
```

Initialize monitor paths.

Search the available monitors and fill up the paths for each monitor.

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.2 Energy Monitor (RAPL MSR)

Functions

- esmi_status_t esmi_core_energy_get (uint32_t core_ind, uint64_t *penergy)

  Get the core energy for a given core.
- esmi_status_t esmi_socket_energy_get (uint32_t socket_ind, uint64_t *penergy)

  Get the socket energy for a given socket.

4.2.1 Detailed Description

Below functions provide interfaces to get the core energy value for a given core and to get the socket energy value for a given socket.

4.2.2 Function Documentation

4.2.2.1 esmi_core_energy_get()

```
esmi_status_t esmi_core_energy_get (
            uint32_t core_ind,
            uint64_t * penergy )
```

Get the core energy for a given core.

Given a core index `core_ind`, and a `penergy` argument for energy profile of that particular cpu, this function will read the energy counter of the given core and update the `peenergy` in micro Joules.

Note: The energy status registers are accessed at core level. In a system with SMT enabled in BIOS, the sibling threads would report duplicate values. Aggregating the energy counters of the sibling threads is incorrect.

Parameters

`in`	*core_ind*	is a core index
`in,out`	*penergy*	The energy profile of a core

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.2.2.2 esmi_socket_energy_get()

```
esmi_status_t esmi_socket_energy_get (
uint32_t socket_ind,
uint64_t * penergy )
```

Get the socket energy for a given socket.

Given a scoket index `socket_ind`, and a `penergy` argument for energy profile of a particular socket. This function identifies an online cpu of the specific socket and reads the socket energy counter.

Updates the `penergy` with socket energy in micro Joules.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*penergy*	The energy profile of a socket

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.3 Power Monitor

Functions

- esmi_status_t esmi_socket_power_avg_get (uint32_t socket_ind, uint32_t ∗ppower)

  Get the average power consumption of the socket with provided socket index.
- esmi_status_t esmi_socket_power_cap_get (uint32_t socket_ind, uint32_t ∗pcap)

  Get the current power cap value for a given socket.
- esmi_status_t esmi_socket_power_cap_max_get (uint32_t socket_ind, uint32_t ∗pmax)

  Get the maximum value that can be assigned as a power cap for a given socket.

4.3.1 Detailed Description

Below functions provide interfaces to get the current power usage and Power Limits for a given socket.

4.3.2 Function Documentation

4.3.2.1 esmi_socket_power_avg_get()

```
esmi_status_t esmi_socket_power_avg_get (
             uint32_t socket_ind,
             uint32_t ∗ ppower )
```

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

Given a socket index `socket_ind` and a pointer to a uint32_t `ppower`, this function will get the current average power consumption (in milliwatts) to the uint32_t pointed to by `ppower`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*ppower*	a pointer to uint32_t to which the average power consumption will get

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.3.2.2 esmi_socket_power_cap_get()

```
esmi_status_t esmi_socket_power_cap_get (
             uint32_t socket_ind,
             uint32_t ∗ pcap )
```

Get the current power cap value for a given socket.

This function will return the valid power cap `pcap` for a given socket @ socket_ind, this value will be used for the system to limit the power.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*pcap*	a pointer to a uint32_t that indicates the valid possible power cap, in milliwatts

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.3.2.3 esmi_socket_power_cap_max_get()

```
esmi_status_t esmi_socket_power_cap_max_get (
              uint32_t socket_ind,
              uint32_t * pmax )
```

Get the maximum value that can be assigned as a power cap for a given socket.

This function will return the maximum possible valid power cap `pmax` from a `socket_ind`.

Parameters

`in`	*socket_ind*	a socket index
`in,out`	*pmax*	a pointer to a uint32_t that indicates the maximum possible power cap, in milliwatts

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.4 Power Control

Functions

- esmi_status_t esmi_socket_power_cap_set (uint32_t socket_ind, uint32_t pcap)

  Set the power cap value for a given socket.

4.4.1 Detailed Description

This function provides a way to control Power Limit.

4.4.2 Function Documentation

4.4.2.1 esmi_socket_power_cap_set()

```
esmi_status_t esmi_socket_power_cap_set (
            uint32_t socket_ind,
            uint32_t pcap )
```

Set the power cap value for a given socket.

This function will set the power cap to the provided value `cap`.

Parameters

`in`	*socket_ind*	a socket index
`in`	*pcap*	a uint32_t that indicates the desired power cap, in milliwatts

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.5 Performance (Boost limit) Monitor

Functions

- esmi_status_t esmi_core_boostlimit_get (uint32_t cpu_ind, uint32_t *pboostlimit)

  Get the boostlimit value for a given core.

4.5.1 Detailed Description

This function provides the current boostlimit value for a given core.

4.5.2 Function Documentation

4.5.2.1 esmi_core_boostlimit_get()

```
esmi_status_t esmi_core_boostlimit_get (
            uint32_t cpu_ind,
            uint32_t * pboostlimit )
```

Get the boostlimit value for a given core.

This function will return the core's current boost limit `pboostlimit` for a particular `cpu_ind`

Parameters

in	*cpu_ind*	a cpu index
in,out	*pboostlimit*	pointer to a uint32_t that indicates the possible boost limit value

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.6 Performance (Boost limit) Control

Functions

- esmi_status_t esmi_core_boostlimit_set (uint32_t cpu_ind, uint32_t boostlimit)

  Set the boostlimit value for a given core.
- esmi_status_t esmi_socket_boostlimit_set (uint32_t socket_ind, uint32_t boostlimit)

  Set the boostlimit value for a given socket.
- esmi_status_t esmi_package_boostlimit_set (uint32_t boostlimit)

  Set the boostlimit value for the whole package (whole system).

4.6.1 Detailed Description

Below functions provide ways to control Boost limit values.

4.6.2 Function Documentation

4.6.2.1 esmi_core_boostlimit_set()

```
esmi_status_t esmi_core_boostlimit_set (
            uint32_t cpu_ind,
            uint32_t boostlimit )
```

Set the boostlimit value for a given core.

This function will set the boostlimit to the provided value `boostlimit` for a given cpu.

Parameters

in	*cpu_ind*	a cpu index is a given core to set the boostlimit
in	*boostlimit*	a uint32_t that indicates the desired boostlimit value of a given core

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.6.2.2 esmi_socket_boostlimit_set()

```
esmi_status_t esmi_socket_boostlimit_set (
            uint32_t socket_ind,
            uint32_t boostlimit )
```

Set the boostlimit value for a given socket.

This function will set the boostlimit to the provided value `boostlimit` for a given socket.

Parameters

in	*socket_ind*	a socket index to set boostlimit
in	*boostlimit*	a uint32_t that indicates the desired boostlimit value of a particular socket

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.6.2.3 esmi_package_boostlimit_set()

```
esmi_status_t esmi_package_boostlimit_set (
            uint32_t boostlimit )
```

Set the boostlimit value for the whole package (whole system).

This function will set the boostlimit to the provided value `boostlimit` for the whole package.

Parameters

in	*boostlimit*	a uint32_t that indicates the desired boostlimit value of the package

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.7 Tctl Monitor

Functions

- esmi_status_t esmi_socket_tctl_get (uint32_t sock_ind, uint32_t ∗ptctl)

  Get the tctl value for a given socket.

4.7.1 Detailed Description

This function provides the current tctl value for a given socket.

4.7.2 Function Documentation

4.7.2.1 esmi_socket_tctl_get()

```
esmi_status_t esmi_socket_tctl_get (
            uint32_t sock_ind,
            uint32_t ∗ ptctl )
```

Get the tctl value for a given socket.

This function will return the socket's current tctl `ptctl` for a particular `sock_ind`

Parameters

`in`	*sock_ind*	a socket index provided.
`in,out`	*ptctl*	pointer to a uint32_t that indicates the possible tctl value

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.8 c0_residency Monitor

Functions

- esmi_status_t esmi_socket_c0_residency_get (uint32_t sock_ind, uint32_t ∗pc0_residency)

  Get the c0_residency value for a given socket.

4.8.1 Detailed Description

This function provides the current c0_residency value for a given socket.

4.8.2 Function Documentation

4.8.2.1 esmi_socket_c0_residency_get()

```
esmi_status_t esmi_socket_c0_residency_get (
            uint32_t sock_ind,
            uint32_t * pc0_residency )
```

Get the c0_residency value for a given socket.

This function will return the socket's current c0_residency `pc0_residency` for a particular `sock_ind`

Parameters

`in`	*sock_ind*	a socket index provided.
`in,out`	*pc0_residency*	pointer to a uint32_t that indicates the possible c0_residency value

Return values

ESMI_SUCCESS	is returned upon successful call.
None-zero	is returned upon failure.

4.9 Auxiliary functions

Functions

- uint32_t esmi_get_number_of_cpus (void)

  Get the number of cpus available.
- uint32_t esmi_get_number_of_sockets (void)

  Get the number of sockets available.
- int esmi_get_online_core_on_socket (int socket_id)

  Get the first online core on a given socket.
- char ∗ esmi_get_err_msg (esmi_status_t esmi_err)

  Get the error string message for esmi errors.

4.9.1 Detailed Description

Below functions provide interfaces to get the total number of cores and sockets available and also to get the first online core on a given socket in the system.

4.9.2 Function Documentation

4.9.2.1 esmi_get_number_of_cpus()

```
uint32_t esmi_get_number_of_cpus (
             void  )
```

Get the number of cpus available.

Get the total number of cpus available in the system

Return values

uint32←_t	is returned upon successful call.

4.9.2.2 esmi_get_number_of_sockets()

```
uint32_t esmi_get_number_of_sockets (
             void  )
```

Get the number of sockets available.

Get the total number of sockets available in the system

Return values

uint32←↩ _t	is returned upon successful call.

4.9.2.3 esmi_get_online_core_on_socket()

```
int esmi_get_online_core_on_socket (
              int socket_id )
```

Get the first online core on a given socket.

Get the online core belongs to particular socket with provided socket index

Parameters

in	*socket←↩ _id*	is a socket index

Return values

int	value returned upon successful call.

4.9.2.4 esmi_get_err_msg()

```
char* esmi_get_err_msg (
              esmi_status_t esmi_err )
```

Get the error string message for esmi errors.

Get the error message for the esmi error numbers

Parameters

in	*esmi_err*	is a esmi error number

Return values

char∗	value returned upon successful call.

Chapter 5

File Documentation

5.1 e_smi.h File Reference

Macros

- #define ENERGY_DEV_NAME "amd_energy"

  Supported Energy driver name.
- #define HSMP_DEV_NAME "amd_hsmp"

  Supported HSMP driver name.
- #define **MAX_CPUS** 1024
- #define **MAX_SOCKETS** 16
- #define FILEPATHSIZ 512

  Buffer to hold size of sysfs filepath.
- #define DRVPATHSIZ 256

  size of driver location path
- #define **FILESIZ** 128
- #define **SYSFS_CPU_PATH** "/sys/devices/system/cpu/present"
- #define **SYSFS_SOCKET_PATH** "/sys/devices/system/node/possible"
- #define HWMON_PATH "/sys/class/hwmon"

  Sysfs directory path for hwmon devices.
- #define CPU_PATH "/sys/devices/system/cpu"

  The core sysfs directory.

Enumerations

- enum esmi_status_t {
  ESMI_SUCCESS = 0, ESMI_INITIALIZED = 0, ESMI_NO_ENERGY_DRV, ESMI_NO_HSMP_DRV,
  ESMI_NO_DRV, ESMI_FILE_NOT_FOUND, ESMI_DEV_BUSY, ESMI_PERMISSION,
  ESMI_NOT_SUPPORTED, ESMI_FILE_ERROR, ESMI_INTERRUPTED, ESMI_UNEXPECTED_SIZE,
  ESMI_UNKNOWN_ERROR, ESMI_ARG_PTR_NULL, ESMI_NO_MEMORY, ESMI_NOT_INITIALIZED,
  ESMI_INVALID_INPUT }

  Error codes retured by E-SMI functions.

Functions

- esmi_status_t esmi_init (void)

  Initialize monitor paths.
- void esmi_exit (void)

  Clean up allocation during init.
- esmi_status_t esmi_core_energy_get (uint32_t core_ind, uint64_t ∗penergy)

  Get the core energy for a given core.
- esmi_status_t esmi_socket_energy_get (uint32_t socket_ind, uint64_t ∗penergy)

  Get the socket energy for a given socket.
- esmi_status_t esmi_socket_power_avg_get (uint32_t socket_ind, uint32_t ∗ppower)

  Get the average power consumption of the socket with provided socket index.
- esmi_status_t esmi_socket_power_cap_get (uint32_t socket_ind, uint32_t ∗pcap)

  Get the current power cap value for a given socket.
- esmi_status_t esmi_socket_power_cap_max_get (uint32_t socket_ind, uint32_t ∗pmax)

  Get the maximum value that can be assigned as a power cap for a given socket.
- esmi_status_t esmi_socket_power_cap_set (uint32_t socket_ind, uint32_t pcap)

  Set the power cap value for a given socket.
- esmi_status_t esmi_core_boostlimit_get (uint32_t cpu_ind, uint32_t ∗pboostlimit)

  Get the boostlimit value for a given core.
- esmi_status_t esmi_core_boostlimit_set (uint32_t cpu_ind, uint32_t boostlimit)

  Set the boostlimit value for a given core.
- esmi_status_t esmi_socket_boostlimit_set (uint32_t socket_ind, uint32_t boostlimit)

  Set the boostlimit value for a given socket.
- esmi_status_t esmi_package_boostlimit_set (uint32_t boostlimit)

  Set the boostlimit value for the whole package (whole system).
- esmi_status_t esmi_socket_tctl_get (uint32_t sock_ind, uint32_t ∗ptctl)

  Get the tctl value for a given socket.
- esmi_status_t esmi_socket_c0_residency_get (uint32_t sock_ind, uint32_t ∗pc0_residency)

  Get the c0_residency value for a given socket.
- uint32_t esmi_get_number_of_cpus (void)

  Get the number of cpus available.
- uint32_t esmi_get_number_of_sockets (void)

  Get the number of sockets available.
- int esmi_get_online_core_on_socket (int socket_id)

  Get the first online core on a given socket.
- char ∗ esmi_get_err_msg (esmi_status_t esmi_err)

  Get the error string message for esmi errors.

5.1.1 Detailed Description

Main header file for the E-SMI library. All required function, structure, enum, etc. definitions should be defined in this file.

This header file contains the following: APIs prototype of the APIs exported by the E-SMI library. Description of the API, arguments and return values. The Error codes returned by the API.

5.1.2 Enumeration Type Documentation

5.1.2.1 esmi_status_t

```
enum esmi_status_t
```

Error codes retured by E-SMI functions.

Enumerator

ESMI_SUCCESS	Operation was successful.
ESMI_INITIALIZED	ESMI initialized successfully.
ESMI_NO_ENERGY_DRV	Energy driver not found.
ESMI_NO_HSMP_DRV	HSMP driver not found.
ESMI_NO_DRV	No Energy and HSMP driver present.
ESMI_FILE_NOT_FOUND	file or directory not found
ESMI_DEV_BUSY	Device or resource busy.
ESMI_PERMISSION	Many functions require root access to run. Permission denied/EACCESS file error.
ESMI_NOT_SUPPORTED	The requested information or action is not available for the given input, on the given system
ESMI_FILE_ERROR	Problem accessing a file. This may because the operation is not supported by the Linux kernel version running on the executing machine
ESMI_INTERRUPTED	execution of function An interrupt occurred during
ESMI_UNEXPECTED_SIZE	was read An unexpected amount of data
ESMI_UNKNOWN_ERROR	An unknown error occurred.
ESMI_ARG_PTR_NULL	Parsed argument is invalid.
ESMI_NO_MEMORY	Not enough memory to allocate.
ESMI_NOT_INITIALIZED	ESMI path not initialized.
ESMI_INVALID_INPUT	Input value is invalid.

Index