extreme computing

HDEEM Library Reference Guide

86 A2 52FP 03 - October 2016

Bull
atos technologies

Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this manual.

All brand names and software and hardware product names are subject to trademark and/or patent protection.

Quoting of brand and product names is for information purposes only and does not represent trademark and/or patent misuse.

Hardware

October 2016

Bull Cedoc
357 avenue Patton
BP 20845
49008 Angers Cedex 01
FRANCE

Table of Contents

Preface

This guide explains how to use the HDEEM library with bullx blades.

Note You are advised to consult the Bull Support Web site for the most up-to-date product information, documentation, firmware updates, software fixes and service offers:
http://support.bull.com

Intended Readers

This guide is intended for HDEEM library users.

Highlighting

The following highlighting conventions are used in this guide:

Bold	Identifies the following: • Interface objects such as menu names, labels, buttons and icons. • File, directory and path names. • Keywords to which particular attention must be paid.
Italic	Identifies references such as manuals or URLs.
`monospace`	Identifies portions of program codes, command lines, or messages displayed in command windows.
< >	Identifies parameters to be supplied by the user.

```
Commands entered by the user
```

```
System messages displayed on the screen
```

Identifies the FRONT of a component.

Identifies the REAR of a component.

DANGER
A Danger notice indicates the presence of a hazard that has the potential of causing death or serious personal injury.

CAUTION
A Caution notice indicates the presence of a hazard that has the potential of causing moderate or minor personal injury.

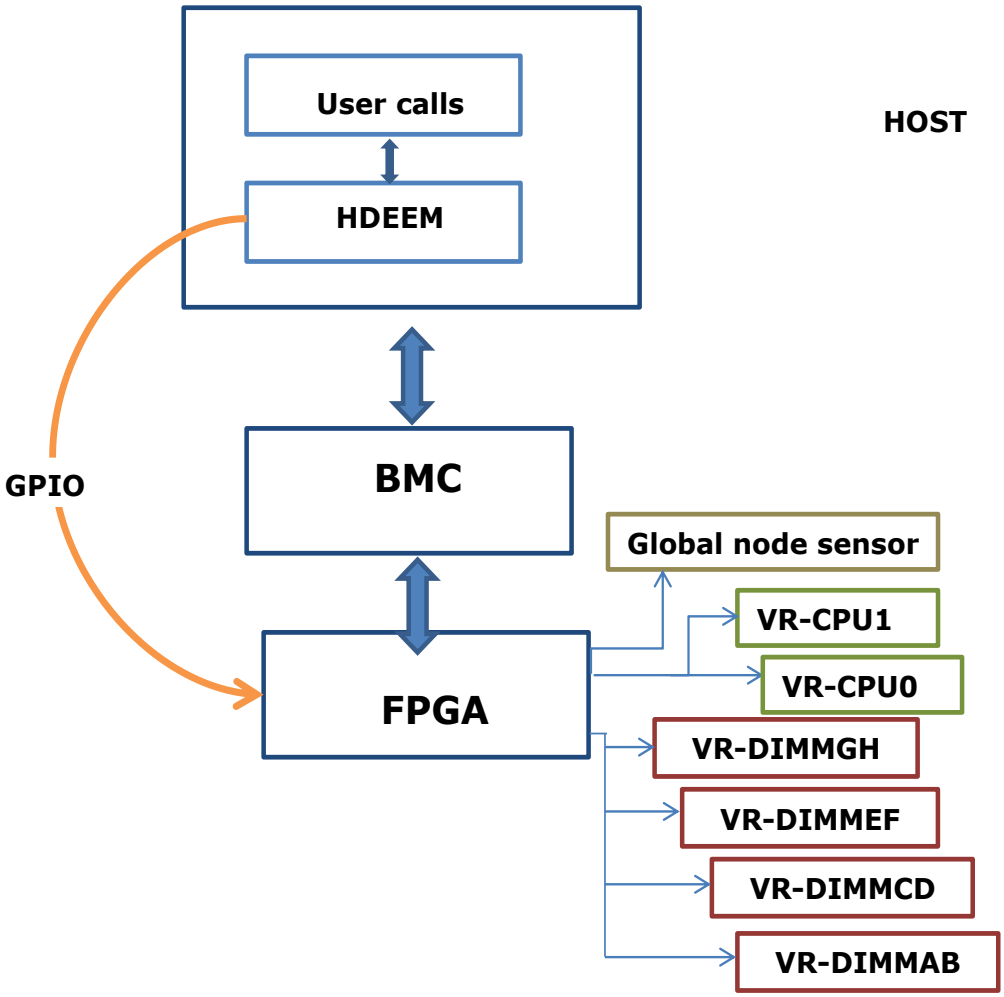
WARNING
A *Warning* notice indicates an action that could cause damage to a program, device, system, or data.

Chapter 1. Overview

The High Definition Energy Efficiency Monitoring (**HDEEM**) library is a software interface used to measure power consumption of HPC clusters with bullx blades.

Measurements are made via the **BMC** (Baseboard Management Controller) and a **FPGA** (Field-Programmable Gate Array) located on each blade.

The diagram, below, shows the hardware and software components for a user application built with the HDEEM interface:



The table below shows the software component stack for the application:

User Application			
HDEEM library			
FreeIPMI		HDEEM_DRV	
LAN	OpenIPMI	PCIe	GPIO
BMC			
FPGA			

The HDEEM library interfaces with the FPGA component either in-band or out-of-band. On out-of-band access, driving the FPGA is done through the BMC which is accessed by the network.

In-band access is done through IPMI, using the OpenIPMI driver for IPMI commands to the BMC. Other accesses need the HDEEM_DRV kernel module to use PCI express data transfers from the FPGA to the BMC, and GPIO signals to start and stop data collection.

A sensor on the overall consumption of the node is read by the FPGA at a frequency of 8000 Hz and filtered values are recorded 1000 times per second.

Six Voltage Regulator sensors corresponding to the VR-CPU0 and VR-CPU1 processor sockets and to the VR-DIMMAB, VR-DIMMCD, VR-DIMMEF and VR-DIMMGH memory channels for each socket, are recorded at a frequency of 100 values per second.

The reading is performed by querying the BMC module which buffers the data using 256 MB of memory amounting to 8 hours of polling from the FPGA.

For a better accuracy of measurement, the blades are calibrated in factory and corrective factors are stored in the firmware of each blade. The error for the blade power values is less than 2%.

The API allows different data uses to be collected:

- Measurement of the cumulated energy, the minimum and the maximum power between a start and a stop command. The time resolution is one millisecond for the blade sensor, and $1/100^{th}$ second for the VR sensors.
- Measurement of the cumulated energy since the power awake of the blade.
- For both cumulated energy counters, from power on or from the start signal, intermediate measurement is accessible with a timestamp.
- Record of instantaneous power values. One value is recorded every millisecond for the blade sensor, and every $1/100^{th}$ second for the VR sensor. Data is stored in a circular buffer in the BMC that holds up to 8 hours of data without being emptied.
- All previous functionalities are available either from the operating system of the system being measured for improved synchronization of the start and stop commands and better response times to empty the circular buffer.
- These are also available through remote access to the BMC, for a null overhead on the host being measured.

1.1 Requisites

The following elements are required to use the HDEEM interface:

- **Red Hat 6.5** to ensure that the GPIO signal, include files and compilers, e.g. **gcc**, can be used.
- Libraries to link applications:
  - **freeipmi-1.2.9-22.1.x86_64.rpm**
  - **freeipmi-devel-1.2.9-22.1.x86_64.rpm**
- The ipmi service to connect to the BMC from the host:
  - **OpenIPMI-2.0.16-14.el6.x86_64.rpm**
  - **OpenIPMI-libs-2.0.16-14.el6.x86_64.rpm**

  With its dependencies:
  - **lm_sensors-libs-3.1.1-17.el6.x86_64.rpm**
  - **net-snmp-libs-5.5-27.el6.x86_64.rpm**

These modules must be installed and the IPMI service must be started using the command:

```
service ipmi start
```

The BMC and FPGA firmware must be at least:

BMC: 40.24.00 build103

FPGA: 0.2.6

Note Earlier BMC versions from 40.20.00 build 87 onwards are supported with limited functionalities.

1.2 Installation

1. Remove previous installations (<2.1.5) using the ./uninstall.sh script.
2. Remove the **/usr/local/hdeem** directory and the **/usr/local/bin/*Hdeem*** binary files
3. Install the rpm:

```
rpm -i hdeem-2.2.1-1.el6.x86_64.rpm
```

with its dependencies on freeipmi and OpenIPMI

1.3 File List

The HDEEM package includes library files and an include file that describes the structure, along with prototypes of the functions used:

The API:

In the **/usr/lib64** directory:

- **libhdeem.a**, the static link version of the HDEEM library
- **libhdeem.so.1.5**, the dynamic version of this library

In **/usr/include**:

- **hdeem.h,** the header file for HDEEM functions

Command line utilities:

In **/usr/bin**:

- **startHdeem** to start high frequency data collection
- **stopHdeem** to stop high frequency data collection
- **checkHdeem** to display the status of the current data collection
- **printHdeem** to dump power values into a .csv file
- **clearHdeem** to reset the status, for example after an overflow.

An example of user code that uses the **hdeem API**:

In **/usr/shared/hdeem/sample**:

- hdeem_sample.c
- Makefile

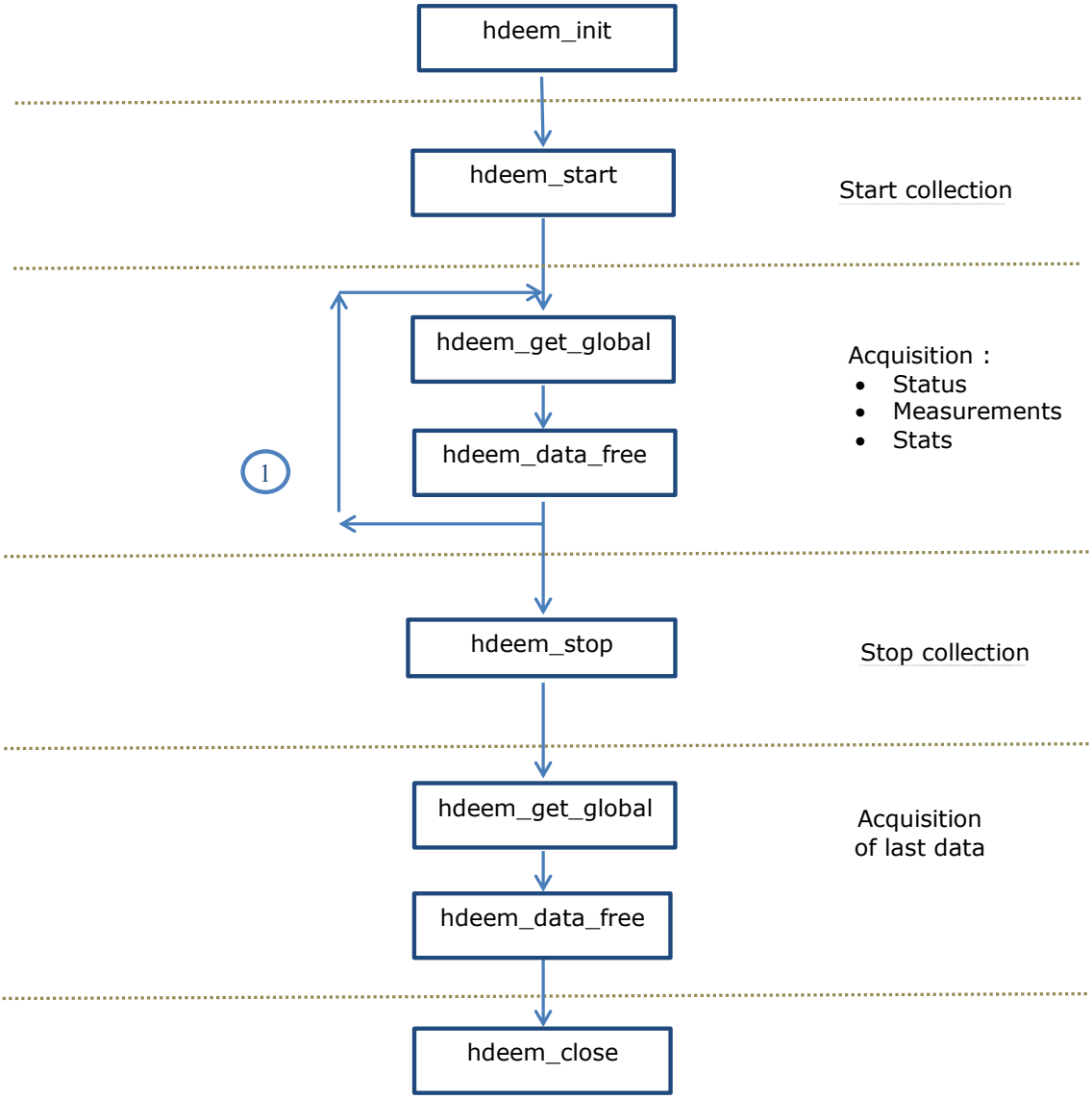
The kernel module to manage access to the BMC

- **/lib/modules /hdeem_drv.ko**
- **/etc/sysconfig/modules/hdeem**.modules
- **/etc/rc.d/init.d/hdeem** (kernel 2.6)
- **/usr/libexec/hdeem** (kernel 3.x)
- **/usr/lib/systemd/system/hdeem.service** (kernel 3.x)

Chapter 2. HDEEM Operation Calls

The HDEEM power measurement tool provides a set of APIs which when executed collect data for an application. The diagram below shows the different call chains for the application:

For data collection:



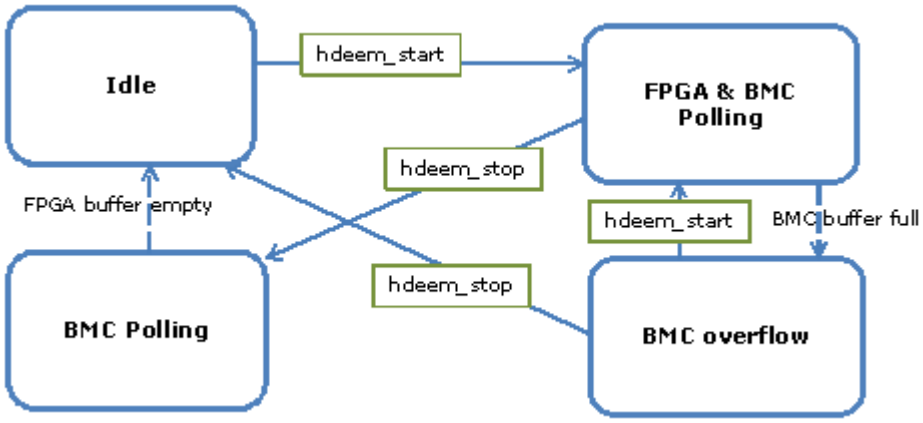
1 Optional intermediate dump of the data. Data collection may not be longer than 8 hours between two calls to hdeem_get_global()

2.1 Operating Principles

- The first call to the HDEEM API must always be **hdeem_init()**
- **hdeem_check_status()**, **hdeem_get_global()** and **hdeem_get_stats()** may be called at any time to get data from the last (possible still ongoing) measurement session
- **hdeem_close()** must be called at the end
- If **hdeem_get_global()** is called **hdeem_data_free()** must be called, otherwise memory leaks arise
- If **hdeem_get_stats()** is called **hdeem_stats_free(**) must be called, otherwise memory leaks arise

2.2 HDEEM System States

The diagram below shows the different system states possible along with the procedure calls used to change application states.



Idle: Sensors are read, calibration is applied, and values are made available to the BMC for its web interface.

FPGA & BMC Polling: Buffer is being filled in FGPA and in BMC and stats (min/max/energy) are gathered in the FPGA

BMC Polling: FPGA polling is stopped, but its buffer has not been emptied by the BMC yet.

FPGA or BMC overflow: Only stats (min/max/energy) are gathered in the FPGA. High resolution data already stored in the BMC remains available for reading.

Chapter 3. Data Structures

The APIs provided by HDEEM use data built according to four data structures:

- **hdeem_bmc_data_t**
- **hdeem_status_t**
- **hdeem_global_reading_t**
- **hdeem_stats_reading_t**

These structures are defined in the **hdeem.h** file and provide information to the application at various levels using fields that are either read or write.

3.1 hdeem_bmc_data_t

This structure applies to the HDEEM connection and its values are obtained during the initialization phase.

```
typedef struct hdeem_bmc_data {
    char* host;
    char *user;
    char* password;
    int hasGPIO ;
    int hasPCIe ;
    ipmi_ctx_t ctx;
    int nb_vr_sensors;
    int nb_blade_sensors;
    char ** name_vr_sensors;
    char ** name_blade_sensors;
    int structure_version ;
    int blade_frequency ;
    int vr_frequency ;
    struct timespec skew_blade;
    struct timespec skew_vr;
} hdeem_bmc_data_t;
```

Member	Description
host	Host name pointer not null for a valid name for out-of-band processing null or empty string for in band processing
user	user name, need if host name is valid
password	user password, need if host name is valid
hasGPIO	Equal to 1: data collection started by GPIO Not equal to 1: data collection stated by IPMI
hasPCIe	Equal to 1: hdeem has PCIe capability Not equal to 1: hdeem does not have PCIe capability
ctx	ipmi connection pointer
nb_vr_sensors	Number of VR sensors
nb_blade_sensors	Number of blade sensors
name_vr_sensors	{ “CPU0”, “CPU1”, “DDR AB”, “DDR CD”, “DDR EF”, “DDR GH”}
name_blade_sensors	{“blade”}
structure_version	Capability of the BMC (1 or 2, version 2 supports instant values and timestamps returned in hdeem_stats_reading)
blade_frequency	Number of samples stored per second for the “blade“ table
vr_frequency	Number of samples stored per second for the “VR“ table
skew_blade	Time between sensor reading at hardware level, and the time it is written in the FPGA table, for the blade sensors
skew_vr	Time between sensor reading at hardware level, and the time it is written in the FPGA table, for the VR sensors

All the fields are read by user except **host**, **user** and **password** which use the values that the user enters.

3.2 hdeem_status_t

This structure provides HDEEM application status details at the time it is called:

```
typedef struct hdeem_status {
    unsigned char status;
    struct timespec start_time_blade;
    struct timespec start_time_vr;
    struct timespec stop_time_blade;
    struct timespec stop_time_vr;
    uint64_t  total_blade_values;
    uint64_t total_vr_values;
    uint64_t pending_blade_values;
    uint64_t pending_vr_values;
} hdeem_status_t;
```

All the fields are read by user.

<table>
<tr><th>Member</th><th>Description</th></tr>
<tr><td>Status</td><td>
Status of BMC or FPGA, a byte with bits signification :
<table>
<tr><th>BIT</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>Desc.</td><td>FPGA polling state</td><td>Polling request source</td><td>Appli. upload. state</td><td>BMC polling FPGA</td><td>BMC memory error</td><td>FPGA blade memory error</td><td>FPGA vr memory error</td><td>Internal use</td></tr>
</table>
Macros are define to test each bit of the status register:
<table>
<tr><th>Macro</th><th>Bit description</th></tr>
<tr><td>IsFpgaPolling(status)</td><td>FPGA polling state :
0 : FPGA is not polling power components, is stopped
1 : FPGA is polling power components</td></tr>
<tr><td>IsStartedByIpmi(status)</td><td>Polling request source – meaningless when data collection is inactive (stopped) :
0 : Measurement session started by GPIO.
1 : Measurement session started by IPMI.</td></tr>
<tr><td>IsUploadingSessionActive(status)</td><td>Application uploading state :
0 : no active upload session
1 : upload session is active</td></tr>
<tr><td>IsBmcPolling(status)</td><td>BMC polling FPGA :
0 : BMC is not polling FPGA
1 : BMC is polling FPGA</td></tr>
<tr><td>IsBmcOverflow(status)</td><td>BMC memory error :
0 : No BMC space memory error
1 : BMC space memory is overflow, so BMC no more polls FPGA</td></tr>
<tr><td>IsFpgaBladeOverflow(status)</td><td>FPGA blade memory error
0 : No FPGA blade space memory error
1 : FPGA blade space memory is overflow, BMC no more polls FPGA, BMC is probably in internal error</td></tr>
<tr><td>IsFpgaVrOverflow(status)</td><td>FPGA VR memory error
0 : No FPGA VR space memory error
1 : FPGA VR space memory is overflow, BMC no more polls FPGA, BMC is probably in internal error</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Member</th><th>Description</th></tr>
<tr><td>Status</td><td>The states in the state diagram correspond to the following bit configurations.
<table>
<tr><th>State</th><th>FPGA Polling</th><th>BMC Polling</th><th>BMC Memory Error</th><th>FPGA Blade Memory Error</th></tr>
<tr><td>Idle</td><td>0</td><td>0</td><td>x</td><td>x</td></tr>
<tr><td>BMC Polling</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>BMC Overflow</td><td>1</td><td>0</td><td>1</td><td>x</td></tr>
<tr><td>FPGA & BMC Polling</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
</table>
Other cases should not occur</td></tr>
<tr><td>start_time_blade</td><td>Time when the first item in the blade table was read. Expressed since Linux epoch (January 1st, 1970).</td></tr>
<tr><td>start_time_vr</td><td>Time when the first item in the VR table was read. Expressed since Linux epoch (January 1st, 1970).</td></tr>
<tr><td>stop_time_blade</td><td>Time when the last item in the blade table was read. Expressed since Linux epoch (January 1st, 1970).</td></tr>
<tr><td>stop_time_blade</td><td>Time when the last item in the VR table was read. Expressed since Linux epoch (January 1st, 1970).</td></tr>
<tr><td>total_blade_values</td><td>Number of blade values.</td></tr>
<tr><td>total_vr_values</td><td>Number of VR values.</td></tr>
<tr><td>pending_blade_values</td><td>Number of blade values pending in BMC.</td></tr>
<tr><td>pending _vr_values</td><td>Number of VR values pending in BMC.</td></tr>
</table>

3.3 hdeem_global_reading_t

This structure is constructed during the call to hdeem_get_global(). It should be freed by a call to hdeem_data_free() after use.

```
typedef struct hdeem_global_reading {
    uint64_t first_blade_value;
    uint64_t first_vr_value;
    uint64_t nb_blade_values;
    uint64_t nb_vr_values;
    hdeem_blade_power_t * blade_power;
    hdeem_vr_power_t * vr_power;
} hdeem_global_reading_t;
```

All the fields are read by user.

Member	Description
first_blade_value	The ordering number of the first blade value returned in this structure numbered since the start of data collection
first_vr_value	The ordering number of the first VR value returned in this structure numbered since the start of data collection
nb_blade_values	Number of **struct hdeem_blade_power** values read in the BMC
nb_vr_values	Number of **struct blade_vr_power** values read in the BMC
blade_power	Pointer of array of blade power values
vr_power	Pointer of array of VR power values

```
typedef struct hdeem_blade_power {
    float *value;                    blade sensor value
} hdeem_blade_power_t;

typedef struct hdeem_vr_power {
    float *value;                    vr sensor values
} hdeem_vr_power_t;

typedef struct hdeem_energy {
    double *value;                   cumulated energy
} hdeem_energy_t;
```

3.4 hdeem_stats_reading_t

This structure is constructed during the call to **hdeem_get_stats()** or **hdeem_get_stats_total()**. It should be freed by a call to **hdeem_stats_free()** after use.

```
typedef struct hdeem_stats_reading {
    hdeem_vr_power_t max_vr_values;
    hdeem_blade_power_t max_blade_values;
    hdeem_vr_power_t min_vr_values;
    hdeem_blade_power_t min_blade_values;
    hdeem_energy_t energy_vr_values;
    hdeem_energy_t energy_blade_values;
    hdeem_vr_power_t average_vr_values;
    hdeem_blade_power_t average_blade_values;
    uint64_t nb_blade_values;
    uint64_t nb_vr_values;

    hdeem_vr_power_t instant_vr_values;
    hdeem_blade_power_t instant_blade_values;

    struct timespec read_time_blade;
    struct timespec read_time_vr;
} hdeem_stats_reading_t;
```

All the fields are read by user.

Members	Description	Unit
max_vr_values	Maximum of VR power sensor values	Watt
max_blade_values	Maximum of blade power sensor values	Watt
min_vr_values	Minimum of VR power sensor values	Watt
min_blade_values	Minimum of blade power sensor values	Watt
energy_vr_values	Energy consumption measured by the VR sensors	Joule
energy_blade_values	Energy consumption measured by the blade sensor	Joule
average_vr_values	Average of VR power sensor values	Watt
average_blade_values	Average of blade power sensor values	Watt
nb_blade_values	Number of samples used to compute statistics about blade power	-
nb_vr_values	Number of samples used to compute statistics about VR power	-
instant_vr_values	Last VR values available	Watt
instant_blade_values	Last blade values available	Watt
read_time_blade	Time when the blade sensors used in this structure were last read	Seconds/Nanoseconds Since 1/1/1970
read_time_vr	Time when the VR sensors used in this structure were last read	Seconds/Nanoseconds Since 1/1/1970

Note The **read_time_blade** and r**ead_time_vr** differ because of the lapse of time between the sensor polling and the time the data is made available after filtering.

Chapter 4. APIs

This chapter lists HDEEM functions with their return codes and examples of use. **hdeem.h** requires **struct timespec**. The easiest way to ensure this is to include **hdeem.h** as the first header. Otherwise the source needs to be compiled according to POSIX P1003.1b-1993 or as ISOC11. Anyone of the following works with glibc/gcc:

- -D_DEFAULT_SOURCE
- -D_POSIX_C_SOURCE=199309L
- -std=gnu99
- -std=c11

4.1 hdeem_init

The **int hdeem_init (hdeem_bmc_data_t * bmc)** function generates a BMC application-specific context and prepares GPIO signal use and initializes the numbers of VR and blade sensors.

This function must be called before any other HDEEM functions.

Parameters
bmc -> host: must be initialized to a valid ip address of the BMC, the hostname of the BMC, or the NULL pointer.
bmc -> user: if host is different from NULL, must be initialized to a valid login name on the BMC.
bmc -> password: if user is different from NULL, must contain the password.

hdeem_init() checks for the compatibility of the **hdeem.h** header used at compile time, and of the dynamic library. When the library is not compatible, an error code is returned. If the library is compatible but the version number is different, a warning is printed.

4.1.1 Return Codes

Return code	Description
0	OK
HDEEM_ERROR_GPIO_INIT	GPIO not available
HDEEM_ERROR_INVALID_PARAMETER	Invalid parameter
HDEEM_ERROR_IPMI_CONNECT	Cannot establish ipmi connection
HDEEM_INTERNAL_ERROR	HDEEM internal error
HDEEM_BMC_INTERNAL_ERROR	BMC internal error
HDEEM_VERSION_MISMATCH	Header and library versions do not match

4.1.2 Example

```
int rc;
...

hdeem_bmc_data_t bmc;
bmc.host = NULL;
...
rc = hdeem_init(&bmc);
if(rc){
        printf("hdeem_init failed!\n");
        return rc;
}
```

4.2 hdeem_close

The **void hdeem_close (hdeem_bmc_data_t * bmc)** function closes and destroys the BMC application-specific context generated by **hdeem_init**.

This function must be called when all other HDEEM functions have completed.

4.2.1 Return Code

Nothing

4.2.2 Example

```
hdeem_bmc_data_t bmc;
int rc;
...
bmc.host = NULL;
rc = hdeem_init(&bmc);
if(rc){
        printf("hdeem_init failed!\n");
        return rc;
}
...
hdeem_close(&bmc);
```

4.3 hdeem_start

The **int hdeem_start (hdeem_bmc_data_t * bmc)** function starts collections at both BMC and FPGA levels and the FPGA receives a GPIO start signal at this time.

4.3.1 Return Code

Return code	Description
0	OK
HDEEM_ERROR_GPIO_ALREADY_START	Data collection already started
HDEEM_ERROR_GPIO_CANNOT_START	Data collection cannot be started by GPIO
HDEEM_ERROR_ALREADY_RUNNING	Data collection already running
HDEEM_ERROR_COMMAND_CONFLICT	Data uploading already running
HDEEM_FPGA_HARDWARE_ERROR_OR_POLLING_STOPPED	Time out with status != 0
HDEEM_ERROR_BMC_POLLING_STILL_RUNNING	Try to start by GPIO but BMC polling is still running
HDEEM_INTERNAL_ERROR	Hdeem internal error
HDEEM_BMC_INTERNAL_ERROR	BMC internal error

4.3.2 Example

```
hdeem_bmc_data_t bmc;
int rc;
...
bmc.host = NULL;
rc = hdeem_init(&bmc);
if(rc){
        printf("hdeem_init failed!\n");
        return rc;
}
...
if(hdeem_start(&bmc)){
        printf("first hdeem_start failed, try hdeem_clear!\n");
        rc = hdeem_clear(&bmc);
        if(rc){
                printf("hdeem_clear failed!\n");
                return rc;
        }
        rc = hdeem_start(&bmc);
        if(rc){
                printf("hdeem_start failed!\n");
                return rc;
        }
}
...
hdeem_close(&bmc);
```

4.4 hdeem_stop

The **int hdeem_stop (hdeem_bmc_data_t * bmc)** function stops the collection at the FPGA level.

It may take some more milliseconds before the polling of the BMC to the FPGA is done and all the data is available for the **hdeem_get_stats** and **hdeem_get_globa**l functions. To be sure, check the BmcPolling bit returned by the **hdeem_check_status** function.

4.4.1 Return Codes

Return code	Description
0	OK
HDEEM_INTERNAL_ERROR	Hdeem internal error
HDEEM_BMC_INTERNAL_ERROR	BMC internal error
HDEEM_ERROR_GPIO_ALREADY_STOP	Data collection already stopped
HDEEM_ERROR_GPIO_CANNOT_STOP	Data collection cannot be stopped by GPIO
HDEEM_ERROR_IPMI_CANNOT_STOP	Data collection cannot be stopped by ipmi
HDEEM_ERROR_BAD_COMMAND	Data collection not running or was not started by ipmi
HDEEM_ERROR_ALREADY_STOPPED	Data collection already stopped
HDEEM_FPGA_HARDWARE_ERROR_OR_POLLING_STOPPED	Time out with status != 0

4.4.2 Example

```
int rc;
...
hdeem_bmc_data_t bmc;
...
bmc.host = NULL;
rc = hdeem_stop(&bmc);
if(rc){
        printf("hdeem_stop failed!\n");
        return rc;
}
...
hdeem_close(&bmc);
```

4.5 hdeem_check_status

The **int hdeem_check_status(hdeem_bmc_data_t * bmc, hdeem_status_t *status)** function retrieves the HDEEM status for an application using a hdeem_status_t data structure call.

4.5.1 Return Code

Return code	Description
0	OK
HDEEM_ERROR_IPMI_CONNECT	Cannot establish ipmi connection
HDEEM_ERROR_BAD_OUTPUT_STRUCTURE_VERSION	Non-supported output structure version
HDEEM_INTERNAL_ERROR	Hdeem internal error
HDEEM_BMC_INTERNAL_ERROR	BMC internal error

4.5.2 Example

```
struct timespec duration, now;
char timestr[30];
int rc;
...
hdeem_bmc_data_t bmc;
hdeem_status_t readings;
...
bmc.host=NULL;
rc = hdeem_init(&bmc);
if(rc){
        printf("hdeem_init failed!\n");
        return rc;
}

rc = hdeem_check_status(&bmc, &readings);
if(rc){
        printf("hdeem_check_status failed!\n");
        return rc;
}
...
printf("FPGA polling         : %d\n", IsFpgaPolling(readStatus.status));
    if (IsFpgaPolling(readStatus.status)) {
        printf("started by           : %s\n",
StartedBy(readStatus.status));
    }
printf("Uploading session    : %d\n",
IsUploadingSessionActive(readStatus.status));
printf("BMC polling          : %d\n", IsBmcPolling(readStatus.status));
printf("BMC overflow         : %d\n", IsBmcOverflow(readStatus.status));
printf("FPGA blade overflow  : %d\n",
IsFpgaBladeOverflow(readStatus.status));
printf("FPGA vr overflow     : %d\n",
IsFpgaVrOverflow(readStatus.status));
printf("Total blade values   : %d\n", readStatus.total_blade_values);
printf("Pending blade in BMC : %d\n", readStatus.pending_blade_values);
printf("Total VR values      : %d\n", readStatus.total_vr_values);
printf("Pending VR in BMC    : %d\n", readStatus.pending_vr_values);
...
hdeem_close(&bmc);
```

4.6 hdeem_get_global

The **int hdeem_get_global(hdeem_bmc_data_t * bmc, hdeem_global_reading_t * readings)** function gets all the sensor values collected by the BMC from the **hdeem_global_reading_t** data structure that have not yet been read by the application at call time.

This function empties memory dedicated to the collection of sensor measurements and should be called when the memory is saturated. The BMC buffering capacity is equivalent to around eight hours of measurements.

This function also allocates memory for the blade and VR value arrays. These two pointers must be freed by calling **hdeem_data_free(),**, even when **hdeem_get_global()** returns an error code.

4.6.1 Return Code

Return code	Description
0	OK
HDEEM_ERROR_IPMI_CONNECT	Cannot establish ipmi connection
HDEEM_ERROR_NOUPLOAD_SESSION_ACTIVE	No upload session active
HDEEM_BMC_MEMORY_FULL	BMC memory is full and cannot collect more data
HDEEM_FPGA_OVERFLOW_GLOBAL	FPGA blade memory is full and cannot collect any more blade data
HDEEM_FPGA_OVERFLOW_VR	FPGA VR memory is full and cannot collect anymore VR data
HDEEM_ERROR_UNABLE_ALLOCATE_MEMORY	Unable to allocate memory
HDEEM_ERROR_START_UPLOAD_CONFLICT	Start upload conflict
HDEEM_ERROR_INVALID_RESERVATION_IDENTIFIER	Invalid reservation identifier
HDEEM_INTERNAL_ERROR	Hdeem internal error
HDEEM_BMC_INTERNAL_ERROR	BMC internal error
HDEEM_ERROR_DATALOST_BYBMC	Data lost by BMC during current or last collection
HDEEM_ERROR_BAD_BLADE_SAMPLE_SIZE	Bad blade sample size
HDEEM_ERROR_BAD_VR_SAMPLE_SIZE	Bad VR sample size

4.6.2 Example

```
int rc;
...
hdeem_bmc_data_t bmc;
hdeem_global_reading_t readings;
...
bmc.host=NULL;
rc = hdeem_init(&bmc);
...
hdeem_start(&bmc);
...
(run the code you want to monitor)
...
rc = hdeem_get_global (&bmc, &readings);
if(rc){
        printf("hdeem_get_global failed!\n");
        hdeem_data_free(&readings);
        return rc;
}
...
    printf("\n%s\n", bmc.name_blade_sensors[0]);
    printf("----------");

    // blade_power contains only one value for the “BLADE” sensor
    for (i = 0; i < readings.nb_blade_values; i++) {
        printf("\n%6d : ",i + readings.first_blade_value);
        printf("  %9.3f", readings.blade_power[i].value[0]);
    }
    printf("\n\n");
    printf("CPUs     \n");
    printf("----------\n");

    // Print the first 2 VR sensors (CPU0 and CPU1)
    for (i = 0; i < readings.nb_vr_values; i++) {
        printf("\n%6d : %12.1f  %12.1f ",i + readings.first_vr_value,
            readings.vr_power[i].value[0],
            readings.vr_power[i].value[1]
              );
    }
...
hdeem_data_free(&readings);
...
hdeem_close(&bmc);
```

4.7 hdeem_get_stats

The **int hdeem_get_stats(hdeem_bmc_data_t * bmc, hdeem_stats_reading_t * stats)** function retrieves statistics obtained by a call, from the beginning of the collection, via a **hdeem_stats_reading_t** data structure.

4.7.1 Return

Return code	Description
0	OK
HDEEM_ERROR_IPMI_CONNECT	Cannot establish ipmi connection
HDEEM_ERROR_BAD_OUTPUT_STRUCTURE_VERSION	Non-supported output structure version
HDEEM_ERROR_BAD_COUNT_STRUCTURES	Structures count not equal to the sum of nb_blade_sensors and nb_vr_sensors
HDEEM_ERROR_BAD_SIZE_STRUCTURE	Structure size equal to 0

4.7.2 Example

```
int rc;
hdeem_stats_reading_t stats;
hdeem_bmc_data_t bmc
...
// Create an connection to the local BMC
bmc.host=NULL;
rc = hdeem_init(&bmc);
if(rc){
        printf("hdeem_init failed!\n");
        return rc;
}
...
// Read statistics about energy consumptions
rc = hdeem_get_stats(&bmc, &stats);

if(rc){
        printf("hdeem_get_stats failed!\n");
        hdeem_stats_free(&stats);
        return rc;
}

printf("\n");
printf("==== HDEEM statistics ====\n");
printf ("            Max (W)        Min (W)        Instant    Average (W)
Energy (J)\n");
for (i = 0; i < bmc.nb_blade_sensors; i++){
    printf("%8s", bmc.name_blade_sensors[i]);
    printf("   %8.3f", stats.max_blade_values.value[i]);
    printf("       %8.3f", stats.min_blade_values.value[i]);
    printf("       %8.3f", stats.instant_blade_values.value[i]);
    printf("       %8.3f", stats.average_blade_values.value[i]);
    printf("      %12.3f\n", stats.energy_blade_values.value[i]);
}
printf("\n");

printf ("            Max (W)        Min (W)        Instant    Average (W)
Energy (J)\n");
for (i = 0; i < bmc.nb_vr_sensors; i++){
    printf("%8s", bmc.name_vr_sensors[i]);
    printf("   %8.3f", stats.max_vr_values.value[i]);
    printf("       %8.3f", stats.min_vr_values.value[i]);
    printf("       %8.3f", stats.instant_vr_values.value[i]);
    printf("       %8.3f", stats.average_vr_values.value[i]);
    printf("      %12.3f\n", stats.energy_vr_values.value[i]);
}
printf("\n");

fflush(stdout);
...
// Free data structures
hdeem_stats_free(&stats);

// Close connection to the BMC
hdeem_close(&bmc);
```

4.8 hdeem_get_stats_total

The **int hdeem_get_stats_total(hdeem_bmc_data_t * bmc, hdeem_stats_reading_t * stats)** function:

- Retrieves the statistics obtained at the time of the call, since the power awake of the blade.
- Returns a hdeem_stats_reading_t structure, same as hdeem_get_stats.

The **max_blade_values** and **max_vr_values** tables are set to a non significant zero.

Return

Return code	Description
0	OK
HDEEM_ERROR_IPMI_CONNECT	Cannot establish ipmi connection
HDEEM_ERROR_BAD_OUTPUT_STRUCTURE_VERSION	Non-supported output structure version
HDEEM_ERROR_BAD_COUNT_STRUCTURES	Structures count not equal to the sum of nb_blade_sensors and nb_vr_sensors
HDEEM_ERROR_BAD_SIZE_STRUCTURE	Structure size equal to 0

4.9 hdeem_clear()

The **int hdeem_clear(hdeem_bmc_data_t * bmc)** function stops FPGA data collection and resets the FPGA and BMC buffers.

This function may be used to reset these components and relaunch a collection cleanly if the previous collection did not stop correctly, for example when there is an application abort.

4.9.1 Return

Return code	Description
0	OK
HDEEM_ERROR_IPMI_CONNECT	Cannot establish ipmi connection
HDEEM_ERROR_NOUPLOAD_SESSION_ACTIVE	No upload session active
HDEEM_ERROR_GPIO_ALREADY_STOP	Data collection already stopped
HDEEM_ERROR_GPIO_CANNOT_STOP	Data collection cannot be stopped by GPIO

4.9.2 Example

```
int rc;
hdeem_bmc_data_t bmc;
...
Bmc.host=NULL;
rc = hdeem_init(&bmc);
if(rc){
        printf("hdeem_init failed!\n");
        return rc;
}
…
if(hdeem_start(&bmc)){
        printf("first hdeem_start failed, try hdeem_clear()!\n");
        rc = hdeem_clear(&bmc);
        if(rc){
                printf("hdeem_clear failed!\n");
                return rc;
        }
        rc = hdeem_start(&bmc);
        if(rc = hdeem_start(&bmc)){
                printf("hdeem_start failed!\n");
                return rc;
        }
}
…
hdeem_close(&bmc);
```

4.10 hdeem_data_free()

void hdeem_data_free(hdeem_global_reading_t * readings)

This function frees memory allocated to the **blade_power** and **vr_power** arrays by the **hdeem_get_global()** function.

This function must follow all **hdeem_get_global()** calls to avoid a memory leak when calling hdeem_get_global()several times.

hdeem_data_free() should be called, even in the event of an error. If the error occurred before the memory was allocated, the corresponding pointers are at least initialized to NULL and **hdeem_data_free()** will not get into segmentation violation.

4.10.1 Return Code

Nothing

4.10.2 Example

```
int rc;
...
hdeem_bmc_data_t * bmc;
hdeem_global_reading_t readings;
...
bmc.host=NULL;
rc = hdeem_init(&bmc);
...
hdeem_start(&bmc);
...
rc = hdeem_get_global (&bmc, &readings);
...
(print many interesting things found in reading)
...
// Free some MBs allocated to contain power values
hdeem_data_free(&readings);
...
hdeem_close(&bmc);
```

4.11 hdeem_stats_free()

The **void hdeem_stats_free(hdeem_stats_reading_t * stats)** function frees memory allocated to arrays by the **hdeem_get_stats()** function.

This function must follow all **hdeem_get_stats()** calls.

4.11.1 Return Code

Nothing

4.11.2 Example

```
int rc;
hdeem_bmc_data_t bmc;
bmc.host=NULL;
hdeem_stats_reading_t stats;
...
// Loop on hdeem_get_stats() to trace the evolution of energy consumed
while (i<1000) {
rc = hdeem_get_stats(&bmc, &stats);

printf("%12.3\n", stats.energy_blade_values.value[0]);

// Avoid a memory leak on the different value[] tables
// allocated at each call to hdeem_get_stats()
hdeem_stats_free(&stats);
}
```

4.12 hdeem_version()

The **void hdeem_version(char * str, int n)** function returns the library version in a string of at most n characters.

4.12.1 Return Code

Nothing.

Printf a warning in stderr if library version differs from user's hdeem.h version.

4.12.2 Example

```
hdeem_version(version, 10);
printf("\nHDEEM_VERSION %s\n\n", version);
```

Chapter 5. Command Line APIs

5.1 startHdeem

Starts in-band or out-of-band collection.

./startHdeem [-H <bmc>] [-U <user>] [-P <password>]: Start high frequency power measurement

<bmc> : IP address or DNS name of the remote BMC for out-of-band access. Default is in-band access

<user> : User name to connect to the remote BMC for out-of-band access

<password>: Password to connect to the remote BMC for out-of-band access

5.2 stopHdeem

Stops in-band or out-of-band data collection.

./stopHdeem [-H <bmc>] [-U <user>] [-P <password>]: Stop high frequency power measurements.

<bmc> : IP address or DNS name of the remote BMC for out-of-band access. Default is in-band access

<user> : User name to connect to the remote BMC for out-of-band access

<password>: Password to connect to the remote BMC for out-of-band access

5.3 checkHdeem

Checks data collection status and displays statistics for the data collected.

./checkHdeem [-H <bmc> -U <user> -P <password>]: Display status of high frequency power measurements.

<bmc> : IP address or DNS name of the remote BMC for out-of-band access. Default is in-band access.

<user> : User name to connect to the remote BMC for out-of-band access.

<password>: Password to connect to the remote BMC for out-of-band access.

Output Example

```
[<user_name>@SP141400Y test_hdeem]$ checkHdeem

==== HDEEM_VERSION 2.2.2 ====

BMC structure version    : 0x2
Synchro capability       : GPIO
Buffer dump through      : PCIe
Blade sensors
     Number of sensors   : 1
     Names               : BLADE,
     Clock skew          : 5.600 ms
     Frequency           : 1000
VR   sensors
     Number of sensors   : 6
     Names               : CPU0, CPU1, DDR_AB, DDR_CD, DDR_EF, DDR_GH,
     Clock skew          : 35.000 ms
     Frequency           : 100
```

```
--------------------------------------------------------------------------------
==== HDEEM status at 2016-05-09 11:01:05.336 ====


HDEEM State       : Idle
BMC polling       : 0
FPGA polling      : 0
Uploading session : 0
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

==== Status of last data collection ====
Last polling start time for blade     : 2016-05-09 10:57:34.618
Last polling start time for vr        : 2016-05-09 10:57:34.589
Polling started by                    : GPIO
Last polling stop time for blade      : 2016-05-09 10:58:39.937
Last polling stop time for VR         : 2016-05-09 10:58:39.908
Polling duration               (s)    : 65.318
BMC overflow                          : 0
FPGA blade overflow                   : 0
FPGA vr overflow                      : 0
Total blade values                    : 65320
Pending blade in BMC                  : 264
Total VR values                       : 6532
Pending VR in BMC                     : 26

==== Instantaneous values ====
    BLADE 101.625
        CPU0  13.875
        CPU1  10.000
    DDR_AB   2.250
    DDR_CD   0.000
    DDR_EF   0.375
    DDR_GH   0.375

==== HDEEM statistics ====
Time of statistics for blade          : 2016-05-09 11:01:05.334
Time of statistics for vr             : 2016-05-09 11:01:05.305
Blade values for stats                : 65319
VR  values for stats                  : 6532
Duration   for stats     (s)          : 65.318
Real blade freq   (#Measure/s)        : 1000.000783
Real vr    freq   (#Measure/s)        : 100.001609

           Max (W)       Min (W)       Average (W)          Energy (J)
BLADE      364.500       142.125       269.405              17597.266
CPU0       119.625       48.500        97.885               6393.865
CPU1       117.625       34.125        93.571               6112.075
DDR_AB     5.375         3.875         5.026                328.305
DDR_CD     3.125         1.625         2.700                176.380
DDR_EF     6.375         0.750         4.619                301.715
DDR_GH     6.250         0.625         4.537                296.353

==== HDEEM statistics total from power awake ====
Time of total stats for blade         : 2016-05-09 11:01:05.336
Time of total stats for vr            : 2016-05-09 11:01:05.307
Blade values total                    : 509455586
VR   values total                     : 50940030

[<user_name>@SP141400Y test_hdeem]$
```

The previous version of the BMC (build 87) is supported with a restricted output:

```
[root@SP141100F ~]# checkHdeem

==== HDEEM_VERSION 2.2.2 ====

BMC structure version    : 0x1
Synchro capability       : GPIO
Buffer dump through      : PCIe
Blade sensors
    Number of sensors    : 1
    Names                : BLADE,
    Clock skew           : 5.600 ms
    Frequency            : 1000
VR  sensors
    Number of sensors    : 6
    Names                : CPU0, CPU1, DDR_AB, DDR_CD, DDR_EF, DDR_GH,
    Clock skew           : 35.000 ms
    Frequency            : 100

==== HDEEM status at 2016-05-09 11:02:54.978 ====
HDEEM State              : Idle
BMC polling              : 0
FPGA polling             : 0
Uploading session        : 0

==== Status of last data collection ====
Last polling start time for blade     : 2016-05-03 16:58:55.467
Last polling start time for vr        : 2016-05-03 16:58:55.437
Polling started by                    : GPIO
Last polling stop time for blade      : 2016-05-03 16:58:56.748
Last polling stop time for VR         : 2016-05-03 16:58:56.718
Polling duration  (s)                 : 1.281
BMC overflow                          : 0
FPGA blade overflow                   : 0
FPGA vr overflow                      : 0
Total blade values                    : 1282
Pending blade in BMC                  : 252
Total VR values                       : 128
Pending VR in BMC                     : 25

==== HDEEM statistics ====
Time of statistics for blade   :      2016-05-09 11:02:54.973
Time of statistics for vr      :      2016-05-09 11:02:54.943
Blade values for stats         :      1281
VR  values for stats           :      128
Duration   for stats     (s)   :      1.281
Real blade freq   (#Measure/s) :      999.949252
Real vr    freq   (#Measure/s) :      99.916865

           Max (W)       Min (W)       Average (W)   Energy (J)
BLADE      92.250        51.750        63.039        80.753
CPU0       27.250        14.125        16.701        21.377
CPU1       22.125         9.250        12.935        16.557
DDR_AB      2.250         0.000         0.399         0.511
DDR_CD      3.375         1.000         1.400         1.792
DDR_EF      3.125         2.250         2.430         3.111
DDR_GH      2.500         1.625         1.801         2.305

hdeem_get_stats_total not supported by this version of BMC
[root@SP141100F ~]#
```

5.4 printHdeem

Prints the values collected into a **csv** file.

./printHdeem [-H <bmc> -U <user> -P <password> -o <file>]: Dump high frequency power value

<bmc> : IP address or DNS name of the remote BMC for out-of-band access. Default is in-band access

<user> : User name to connect to the remote BMC for out-of-band access

<password>: Password to connect to the remote BMC for out-of-band access

<file> : csv output file. Default file name is **hdeem_<date>_<time>.csv**

Example of Generated File

```
HDEEM_VERSION, 2.2.2

BMC address, 172.31.90.80

==== HDEEM status ====              ,
Last start time for blade           , 2016-02-03 11:50:34.784
Last start time for vr              , 2016-02-03 11:50:34.755
Started by                          , IPMI
FPGA polling                        , 0
Last stop time for blade            , 2016-02-03 11:50:35.877
Last stop time for vr               , 2016-02-03 11:50:35.848
Polling duration                (s) , 1.092
Uploading session                   , 0
BMC polling                         , 1
BMC overflow                        , 0
FPGA blade overflow                 , 0
FPGA vr overflow                    , 0
Total blade values                  , 1098
Pending blade in BMC                , 1098
Total VR values                     , 110
Pending VR in BMC                   , 110
----------
               BLADE,
       1,     74.625,
       2,     72.750,
       3,     77.625,
       4,     78.750,
       5,     73.125,
       6,     72.750,
       7,     75.375,
       8,     75.000,
       9,     72.250,
      10,     76.875,
(…)
----------
             CPU0,    CPU1,  DDR_AB,  DDR_CD,  DDR_EF,  DDR_GH,
       1,    14.4,    10.8,     2.2,     0.5,     1.0,     1.0,
       2,    15.0,    11.6,     2.2,     0.5,     1.0,     1.0,
       3,    15.2,    11.6,     2.2,     0.5,     1.0,     1.0,
       4,    15.0,    11.4,     2.2,     0.5,     1.0,     1.0,
       5,    15.0,    11.6,     2.4,     0.5,     1.0,     1.0,
       6,    14.5,    11.0,     2.2,     0.4,     1.0,     1.0,
       7,    14.5,    10.8,     2.2,     0.4,     1.0,     1.0,
       8,    15.4,    12.0,     2.4,     0.8,     1.0,     1.0,
       9,    15.2,    12.0,     2.4,     0.8,     1.0,     1.0,
      10,    14.8,    11.0,     2.2,     0.4,     1.0,     1.0,
```

5.5 clearHdeem

Clears the status of a bogus data collection.

./clearHdeem [-H <bmc>] [-U <user>] [-P <password>]: Display status of high frequency power measurement

<bmc> : IP address or DNS name of the remote BMC for out-of-band access. Default is in-band access

<user> : User name to connect to the remote BMC in case of out-of-band access

<password>: Password to connect to the remote BMC in case of out-of-band access

Chapter 6. Error Codes

The following table summarizes the library function error codes functions and the HDEEM system internal error codes:

Error code	Value	Meaning
HDEEM_FPGA_OVERFLOW_GLOBAL	0x0001	FPGA blade memory is full and cannot collect more blade data
HDEEM_FPGA_OVERFLOW_VR	0x0002	FPGA vr memory is full and cannot collect more blade data
HDEEM_ERROR_PCIE_CONNECT	0x0008	Cannot connect to BMC pilot3, function 1 PCIe interface.
HDEEM_ERROR_GPIO_INIT	0x0010	GPIO not available
HDEEM_ERROR_GPIO_ALREADY_START	0x0015	Data collection already started
HDEEM_ERROR_GPIO_CANNOT_START	0x0016	Data collection cannot be started by GPIO
HDEEM_ERROR_GPIO_ALREADY_STOP	0x0017	Data collection already stopped
HDEEM_ERROR_GPIO_CANNOT_STOP	0x0018	Data collection cannot be stopped by GPIO
HDEEM_ERROR_IPMI_CANNOT_STOP	0x0019	Data collection cannot be stopped by ipmi
HDEEM_ERROR_ALREADY_RUNNING	0x0020	Data collection already running
HDEEM_ERROR_ALREADY_STOPPED	0x0025	Data collection already stopped
HDEEM_ERROR_COMMAND_CONFLICT	0x0030	Data uploading already running
HDEEM_ERROR_BAD_COMMAND	0x0040	Data collection not running or was not started by IPMI
HDEEM_ERROR_INVALID_RESERVATION_IDENTIFIER	0x0050	Invalid reservation identifier
HDEEM_ERROR_DATA_LENGTH_INVALID	0x0060	Internal error in IPMI command
HDEEM_ERROR_OUT_OF_RANGE	0x0070	Internal error in IPMI command
HDEEM_BMC_MEMORY_FULL	0x0078	BMC memory is full and cannot collect more data
HDEEM_FPGA_HARDWARE_ERROR_OR_POLLING_STOPPED	0x0079	Time out with status != 0
HDEEM_BMC_INTERNAL_ERROR	0x0080	BMC internal error
HDEEM_UNKNOWN_ERROR	0x0085	Unknown error
HDEEM_ERROR_IPMI_CONNECT	0x0088	Cannot establish ipmi connection
HDEEM_ERROR_NOUPLOAD_SESSION_ACTIVE	0x0090	No upload session active
HDEEM_ERROR_DATALOST_BYBMC	0x0110	Data lost by BMC during current or last collection
HDEEM_ERROR_START_UPLOAD_CONFLICT	0x0120	Start upload conflict
HDEEM_ERROR_ CONNECT	0x0130	ipmi connection lost
HDEEM_ERROR_UNABLE_ALLOCATE_MEMORY	0x0140	Unable to allocate

Error code	Value	Meaning
		memory
HDEEM_ERROR_INVALID_PARAMETER	0x0145	Invalid parameter
HDEEM_ERROR_BAD_COUNT_STRUCTURES	0x0150	Structures count not equal to the sum of nb_blade_sensors and nb_vr_sensors
HDEEM_ERROR_BAD_SIZE_STRUCTURE	0x0155	Structure size equal to 0
HDEEM_ERROR_BMC_POLLING_STILL_RUNNING	0x0160	Try to start by GPIO but BMC polling is still running
HDEEM_ERROR_BAD_BLADE_SAMPLE_SIZE	0x0165	Bad blade sample size
HDEEM_ERROR_BAD_VR_SAMPLE_SIZE	0x0170	Bad vr sample size
HDEEM_ERROR_BAD_OUTPUT_STRUCTURE_VERSION	0x0175	Non-supported output structure version
HDEEM_ERROR_NO_TIME_FILE	0x0180	Time file not exist
HDEEM_VERSION_MISMATCH	0x0185	Header and library versions do not match
HDEEM_INTERNAL_ERROR	0xFFFF	Hdeem internal error

Chapter 7. Troubleshooting

This chapter lists the preliminary checks to be performed and some potential data collection problems.

7.1 Installation

Check the following before using the HDEEM library:

1. The **hdeem** service is started at boot time, by running:

```
chkconfig --list hdeem
```

If the result is **off**, run:

```
chkconfig hdeem on
```

2. The **hdeem** service has started, by running

```
service hdeem status
```

If the status has stopped, run:

```
service hdeem start
```

3. **hdeem_drv.ko** is inserted as a module (for PCIe capability)

```
lsmod | grep hdeem_drv
```

4. The **ipmi** service is started at boot time, by running:

```
chkconfig --list ipmi
```

If the result is **off**, run:

```
chkconfig ipmi on
```

5. The **hdeem** service has started, by running

```
service hdeem status
```

If the status has stopped, run:

```
service hdeem start
```

7.2 Restart after an Error

The **hdeem_clear** resets the HDEEM system function before restarting a collection properly. This function unblocks potentially critical situations, for example, if a program using the API crashes during the execution of **hdeem_get_global()** and the 'Uploading session' bit is set.

7.3 Communication Failure

If data (blade value and/or vr value) is lost by the BMC, the function **hdeem_get_global** returns the **HDEEM_ERROR_DATALOST_BYBMC** error code.

See *Section 4.6 hdeem_get_global*

The next time **hdeem_get_global** is called, the number of first values read by the second call (**first_blade_value** and **first_vr_value**) may differ from the number of last values returned by the previous call plus 1.

See *Section 4.6 hdeem_get_global* and *Section 3.3 hdeem_global_reading_t*

Bull Cedoc
357 avenue Patton
BP 20845
49008 Angers Cedex 01
FRANCE