CAPMC API Documentation

Release 1.2.2

Cray Inc.

Jul 26, 2018

Revision History

Date	Revision	Summary
Jun 2015	1.0	Initial Release
Dec 2015	1.1	New features and clarifications matching the SMW-8.0 / CLE-6.0 software release
Oct 2016	1.2	Modify 'get_system_parameters' API 'ramp_limited' member data type. Clarify 'node_status –filter' parameter syntax. Clarify '–nids nidlist' parameter syntax in various CLI interface descriptions. Fix 'node_on' refs in 'node_off' example Fix 'json –resource' CLI example Add 'node_reinit' API call Add MCDRAM, NUMA, SSD API calls
Feb 2017	1.2.1	Update to use Intel offical trademark names
Jul 2018	1.2.2	Fix 'node_reinit' API call response description

CONTENTS

1 Capmc CLI **1**
- 1.1 Installation . . . 1
- 1.2 Return Values . . . 1
- 1.3 Credentials . . . 2

2 Node Status & On Off Control **3**
- 2.1 node_rules . . . 3
- 2.2 node_status . . . 5
- 2.3 node_on . . . 7
- 2.4 node_off . . . 8
- 2.5 node_reinit . . . 10

3 Node Capabilities & Power Control **13**
- 3.1 get_power_cap_capabilities . . . 13
- 3.2 get_power_cap . . . 16
- 3.3 set_power_cap . . . 19
- 3.4 get_power_bias . . . 21
- 3.5 set_power_bias . . . 23
- 3.6 clr_power_bias . . . 24
- 3.7 set_power_bias_data . . . 25
- 3.8 compute_power_bias . . . 26

4 Node Frequency & Sleep State Control **29**
- 4.1 get_freq_capabilities . . . 29
- 4.2 get_freq_limits . . . 32
- 4.3 set_freq_limits . . . 35
- 4.4 get_sleep_state_limit_capabilities . . . 38
- 4.5 get_sleep_state_limit . . . 40
- 4.6 set_sleep_state_limit . . . 43

5 Node MCDRAM Capabilities & Control **47**
- 5.1 get_mcdram_capabilities . . . 47
- 5.2 get_mcdram_cfg . . . 49
- 5.3 set_mcdram_cfg . . . 52
- 5.4 clr_mcdram_cfg . . . 54

6 Node NUMA Capabilities & Control **57**
- 6.1 get_numa_capabilities . . . 57
- 6.2 get_numa_cfg . . . 59
- 6.3 set_numa_cfg . . . 61
- 6.4 clr_numa_cfg . . . 63

7 Node SSD Control & Diagnostics **67**
7.1 get_ssd_enable . . . 67
7.2 set_ssd_enable . . . 69
7.3 clr_ssd_enable . . . 71
7.4 get_ssds . . . 72
7.5 get_ssd_diags . . . 74

8 Node Energy Reporting **77**
8.1 get_node_energy . . . 77
8.2 get_node_energy_stats . . . 79
8.3 get_node_energy_counter . . . 82

9 System Level Monitoring **85**
9.1 get_system_parameters . . . 85
9.2 get_system_power . . . 87
9.3 get_system_power_details . . . 88

10 Utility Functions **91**
10.1 get_nid_map . . . 91
10.2 get_partition_map . . . 93
10.3 json . . . 94

A Glossary **97**

HTTP Routing Table **99**

Index **101**

CHAPTER

ONE

CAPMC CLI

The capmc utility provides remote monitoring and control capabilities to agents running externally to the Cray System Management Workstation (SMW). The capmc client utility accepts command line arguments from the caller and submits an appropriately formatted request via HTTPS to the monitoring and control service providers running on the SMW. Command results are supplied as JSON-formatted text to standard output. Remote access is restricted by means of public-key authorization established by the site administrator.

In order to use the capmc API calls, also referred to as *applets*, caller must first load the capmc module:

```
$ module load capmc
```

Alternatively, the caller can invoke capmc by specifying the absolute path (determined on installation). The default path is: **/opt/cray/capmc/default/bin/capmc**

1.1 Installation

The capmc utility is packaged in a noarch RPM format. Its only dependency is the Python Standard Library with SSL support enabled, version 2.6 or 2.7. On RPM based systems, the package may be installed with the native package manager.

```
$ rpm -ivh cray-capmc-1.0-1.0000.36550.40.1.noarch.rpm
Preparing...       ########################################### [100%]
   1:cray-capmc    ########################################### [100%]
```

Alternately, capmc may be installed on systems that do not support the RPM format using the rpm2cpio utility. The following example extracts the package contents into the current working directory.

```
$ rpm2cpio cray-capmc-1.0-1.0000.36550.40.1.noarch.rpm | cpio -idmv
./etc/opt/cray
./etc/opt/cray/capmc
./opt/cray
./opt/cray/capmc
./opt/cray/capmc/1.0-1.0000.36550.40.1
./opt/cray/capmc/1.0-1.0000.36550.40.1/bin
./opt/cray/capmc/1.0-1.0000.36550.40.1/bin/capmc
...
```

1.2 Return Values

The capmc utility returns 0 to the shell if the request was successfully submitted to the platform monitoring and control service. Individual command status is indicated in the JSON-encoded output data 'e' member. 'e' is always present in

the response payloads and will be non-zero to indicate a command specific error code if the requested operation failed. Callers of capmc should always evaluate result status returned in the message payload.

1.3 Credentials

Capmc utilizes an X.509 client certificate for authorization. The signed client certificate and private key will be supplied by the system administrator. Credentials may be supplied through environment variables or a configuration file.

1.3.1 Environment Variables

The required environment variables are listed below. Environment variables will override configuration file parameters. Environment variables which must be set to utilize client certificate authorization are as follows:

OS_KEY
: Specify the absolute path in the local filesystem where the X.509 client certificate key is installed. If a pass phrase is set on the key, the user will be prompted when required.

OS_CERT
: Specify the absolute path in the local filesystem where the X.509 client certificate is installed.

OS_CACERT
: Specify the absolute path in the local filesystem where the X.509 Certificate Authority (CA) certificate is located.

OS_SERVICE_URL
: Specify the URL where the application service is listening. This must include the fully qualified domain name (FQDN) of the SMW, protocol, and port number. The default port number is 8443.

1.3.2 Configuration File

When capmc is utilized for autonomous machine to machine communication, it is advisable to define the certificate paths and service url in a configuration file instead of the callers environment. Capmc reads its configuration file from the following location: **/etc/opt/cray/capmc/capmc.json**

X.509 client certificate configuration syntax:

```
{
   "os_key":         "/etc/opt/cray/capmc/capmc-client.key",
   "os_cert":        "/etc/opt/cray/capmc/capmc-client.pem",
   "os_cacert":      "/etc/opt/cray/capmc/capmc-cacert.pem",
   "os_service_url": "https://smw.example.com:8443"
}
```

Note: Capmc validates all X.509 certificates. It is assumed that all host certificates used within HTTPS API servers such as Xtremoted, or X.509 client authorization are signed by the same certificate authority.

Warning: The configuration and X.509 certificate files must have appropriate filesystem permissions. Users with read access to the global configuration file and the client certificate will be granted permission to utilize all capmc functionality.

CHAPTER

TWO

NODE STATUS & ON OFF CONTROL

CAPMC node power controls implement a simple interface for powering on or off compute nodes, querying node state information, and querying site-specific service usage rules. These controls enable external software to more intelligently manage system wide power consumption or configuration parameters. The simplest power management strategy may be to simply turn off compute nodes which may be idle for a significant time interval and turn them back on when demand increases. The following API calls are provided as a means for third party software to implement power management strategies as simple or complex as the site-level requirements demand.

2.1 node_rules

node_rules informs third party software about hardware (and perhaps site-specific) rules or timing constraints that allow for efficient and effective management of idle node resources. The data returned informs the caller of how long **node_on** and **node_off** operations are expected to take, the minimum amount of time nodes should be left off to save energy, and limits on the number of nodes that should be turned on or off at once. Default rules are supplied where appropriate.

Other values such as the maximum node counts for **node_on** or **node_off** and the maximum amount of time a node should remain off after a power down are left unset. The values are not strictly enforced by Cray system management software. They are meant to provide guidelines for authorized callers in their use of the CAPMC service.

2.1.1 CLI Interface

capmc node_rules

The command line interface has no arguments. Result text is returned via standard out.

```
$ capmc node_rules
{
  "e": 0,
  "err_msg": "",
  "latency_node_off": 60,
  ...
```

2.1.2 HTTP Interface

POST /capmc/get_node_rules

Example request:

```
POST /capmc/get_node_rules HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 2
Content-Type: application/json

{}
```

The request must **POST** an empty JSON object to the API server. This command takes no arguments.

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "latency_node_off": 60,
  "latency_node_on": 600,
  "latency_node_reinit": 760,
  "max_off_req_count": -1,
  "max_off_time": -1,
  "max_on_req_count": -1,
  "max_reinit_req_count": -1,
  "min_off_time": 900
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **latency_node_off** (*int*) – Approximate time, in seconds, in which a node is expected to perform a clean shutdown and power off
- **latency_node_on** (*int*) – Approximate time, in seconds, in which a node is expected to perform a warm bounce and boot into a ready state
- **latency_node_reinit** (*int*) – Approximate time, in seconds, in which a node is expected to perform a clean shutdown, power off, warm bounce, and boot into a ready state
- **max_off_req_count** (*int*) – Maximum number of nodes which may be powered off at once. -1 indicates no limit
- **max_off_time** (*int*) – Maximum time, in seconds, in which a node may be in the powered off state. -1 indicates no limit
- **max_on_req_count** (*int*) – Maximum number of nodes which may be powered on and booted at once. -1 indicates no limit
- **max_reinit_req_count** (*int*) – Maximum number of nodes which may be shutdown, powered off, powered on, and rebooted at once. -1 indicates no limit
- **min_off_time** (*int*) – Minimum time, in seconds, in which a node must remain in the powered off state after a shutdown and power off operation. -1 indicates no limit

Status Codes

- 200 OK – Network API call success

Note: Default rules are established automatically at system installation time. The administrator may choose to customize the rule set. Customization is performed by editing the respective parameter in a configuration file (**/opt/cray/hss/default/etc/xtremoted/rules.ini**) stored on the System Management Workstation.

2.2 node_status

A node's component state may be returned via the **node_status** function for the full system or a subset as specified by a nid list or component filter. The status API call is intended, but not limited, to be used in conjunction with asynchronous operations which may modify node component state, such as **node_on** or **node_off**.

Third party utilities would issue an asynchronous operation, such as **node_on**, and if the operation was successfully enqueued poll for changes in component state after waiting for the expected boot time latency. If the targeted component state has switched from "off" to "ready" then the caller knows the operation was successful.

States reported through this API call mirror those defined in Cray HSS.

Node States:

- **disabled** - Component is physically installed, but ignored by Cray system management software
- **halt** - Operating system has shut down, hardware has not yet powered off
- **on** - Power is on and BIOS has initialized all hardware, node is waiting to be booted
- **off** - Power is off
- **ready** - Operating system is fully booted
- **standby** - Operating system is in process of booting

Note: The **node_status** API call does not report **empty** components.

2.2.1 CLI Interface

capmc node_status

-n, --nids <nidlist>
: Specify the NIDs to query status. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

-f, --filter <opts>
: Specify filters for status query. Filters may be pipe-separated (|) and surrounded with double quotes, e.g. "opt|opt|opt". Valid filters are: **show_all**, **show_off**, **show_on**, **show_halt**, **show_standby**, **show_ready**, **show_diag**, and **show_disabled**. If omitted, the default is **show_all**.

```
$ capmc node_status --nids=1,40-43 --filter="show_on|show_ready"
{
    "e": 0,
    "err_msg": "",
    "on": [
        40,
        42,
        43,
```

(continues on next page)

(continued from previous page)

```
        41
    ],
    "ready": [
        1
    ]
...
```

2.2.2 HTTP Interface

POST /capmc/get_node_status

Example Request:

```
POST /capmc/get_node_status HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 96
Content-Type: application/json

{
  "filter": "show_on|show_ready",
  "nids": [
    1,
    40,
    41,
    42,
    43
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes two optional arguments which identify a NID list and component status filter.

JSON Parameters

- **filter** (*string*) – Pipe concatenated (|) list of filter strings
- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "on": [
    40,
    42,
    43,
    41
  ],
  "ready": [
    1
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success, nonzero on invalid input.
- **err_msg** (*string*) – Human readable error message
- **disabled** (*int[]*) – Optional member, list of disabled NIDs
- **halt** (*int[]*) – Optional, list of halted NIDs
- **on** (*int[]*) – Optional, list of powered on NIDs
- **off** (*int[]*) – Optional, list of powered off NIDs
- **ready** (*int[]*) – Optional, list of booted NIDs
- **standby** (*int[]*) – Optional, list of booting NIDs

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

2.3 node_on

Power on and warm boot a selected list of compute node NIDs using the default boot image. This has no effect on the status of the high speed network (HSN). However, this command requires that the HSN ASIC attached to each node in the target list has previously been powered on and routed.

The **node_on** API call is **asynchronous**. It returns immediately containing a status result indicating an error with invalid input parameters, or success indicating the operation has been enqueued into an asynchronous command processing queue. The caller must determine overall command status by polling for **node_status** after the expected power on and warm boot period has elapsed.

2.3.1 CLI Interface

capmc node_on

-n, --nids <nidlist>
: Specify the NIDs to power on and boot. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). Only compute node NIDs may be specified. This is a required option.

-r, --reason <log message>
: Specify an arbitrary text message which is given as the reason for performing the node_on operation. This argument is optional and used for informational purposes when reviewing system log messages. The value is not interpreted in any way by the Cray system management software.

```
$ capmc node_on --nids=40-43 --reason="need more capacity"
{
    "e": 0,
    "err_msg": ""
}
```

2.3.2 HTTP Interface

POST /capmc/node_on

Example Request:

```
POST /capmc/node_on HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 85
Content-Type: application/json

{
  "reason": "need more capacity",
  "nids": [
    40,
    41,
    42,
    43
  ]
}
```

The request must **POST** an array of selected NIDs along with an optional human readable reason string.

JSON Parameters

- **reason** (*string*) – Arbitrary, free-form text
- **nids** (*int[]*) – User specified list of compute node NIDs to warm bounce and boot. An empty array is invalid. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": ""
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success, nonzero on invalid input.
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

2.4 node_off

Shutdown and power off a selected list of compute node NIDs. This has no effect on the status of the high speed network (HSN). The HSN ASIC attached to each node will remain powered on and routed. After the **node_off** operation has completed, the selected nodes will be in a state suitable for warm booting back into the system at a later date.

The **node_off** API call is **asynchronous**. It returns immediately containing a status result indicating an error with invalid input parameters, or success indicating the operation has been enqueued into an asynchronous command processing queue. The caller must determine overall command status by polling for **node_status** after the expected shutdown and power off period has elapsed.

2.4.1 CLI Interface

capmc node_off

-n, --nids <nidlist>
: Specify the NIDs to shut down and power off. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). Only compute node NIDs may be specified. This is a required option.

-r, --reason <log message>
: Specify an arbitrary text message which is given as the reason for performing the node_off operation. This argument is optional and is used in the same manner as with the **node_on** command.

```
$ capmc node_off --nids=40-43 --reason="powersave, need less capacity"
{
    "e": 0,
    "err_msg": ""
}
```

2.4.2 HTTP Interface

POST /capmc/node_off

Example Request:

```
POST /capmc/node_off HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 96
Content-Type: application/json

{
  "reason": "powersave, need less capacity",
  "nids": [
    40,
    41,
    42,
    43
  ]
}
```

The request must **POST** an array of selected NIDs along with an optional human readable reason string.

JSON Parameters

- **reason** (*string*) – Arbitrary, free-form text
- **nids** (*int[]*) – User specified list of compute node NIDs to shutdown and power off. An empty array is invalid. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": ""
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success, nonzero on invalid input.
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

2.5 node_reinit

Warmbounce and boot a selected list of compute node NIDs using the default boot image. This has no effect on the status of the high speed network (HSN). However, this command requires that the HSN ASIC attached to each node in the target list has previously been powered on and routed.

The **node_reinit** API call is **asynchronous**. It returns immediately containing a status result indicating an error with invalid input parameters, or success indicating the operation has been enqueued into an asynchronous command processing queue. The caller must determine overall command status by polling for **node_status** after the expected warmbounce and boot period has elapsed.

2.5.1 CLI Interface

capmc node_reinit

-n, --nids <nidlist>
: Specify the NIDs to warmbounce and boot. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). Only compute node NIDs may be specified. This is a required option.

-r, --reason <log message>
: Specify an arbitrary text message which is given as the reason for performing the node_off operation. This argument is optional and is used in the same manner as with the **node_on** command.

```
$ capmc node_reinit --nids=40-43 --reason="apply staged node configurations"
{
    "e": 0,
    "err_msg": ""
}
```

2.5.2 HTTP Interface

POST /capmc/node_reinit
: **Example Request**:

```
POST /capmc/node_reinit HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 102
Content-Type: application/json

{
  "reason": "apply staged node configurations",
  "nids": [
    40,
    41,
    42,
    43
  ]
}
```

The request must **POST** an array of selected NIDs along with an optional human readable reason string.

JSON Parameters

- **reason** (*string*) – Arbitrary, free-form text
- **nids** (*int[]*) – User specified list of compute node NIDs to shutdown and power off. An empty array is invalid. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": ""
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success, nonzero on invalid input.
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

CHAPTER

THREE

NODE CAPABILITIES & POWER CONTROL

CAPMC power capping controls implement a simple interface for querying component capabilities and manipulation of node or sub-node (accelerator) power constraints. This functionality enables external software to establish an upper bound, or estimate a minimum bound, on the amount of power a system or a select subset of the system may consume. The following API calls are provided as a means for third party software to implement advanced power management strategies.

Several calls do not include a command line interface. A caller may utilize the **capmc json** functionality to send and receive customized JSON data structures conforming to the specification for these calls.

The API calls, **get_power_bias**, **set_power_bias**, **clr_power_bias**, **set_power_bias_data**, & **compute_power_bias**, are new in SMW-8.0.

3.1 get_power_cap_capabilities

The **get_power_cap_capabilities** call informs third-party software about installed hardware and its associated properties. Information returned includes the specific hardware types, NID membership, and power capping controls along with their allowable ranges. Information may be returned for a targeted set of NIDs or the system as a whole.

3.1.1 CLI Interface

capmc get_power_cap_capabilities

-n, --nids <nidlist>
: Specify the NIDs to retrieve power capping capabilities. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs. The NIDs may be in any state.

```
$ capmc get_power_cap_capabilities --nids=40-43,48-49
{
  "e": 0,
  "err_msg": "",
  "groups": [
    {
      "name": "01:000d:206d:0073:0008:0020:3a34:8100",
      "desc": "ComputeANC_SNB_115W_8c_32GB_14900_KeplerK20XAccel",
      "host_limit_max": 185,
 ...
```

3.1.2 HTTP Interface

POST /capmc/get_power_cap_capabilities

Example Request:

```
POST /capmc/get_power_cap_capabilities HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "groups": [
    {
      "name": "01:000d:206d:0073:0008:0020:3a34:8100",
      "desc": "ComputeANC_SNB_115W_8c_32GB_14900_KeplerK20XAccel",
      "host_limit_max": 185,
      "host_limit_min": 95,
      "static": 0,
      "supply": 425,
      "powerup": 120,
      "nids": [
        48,
        49
      ],
      "controls": [
        {
          "name": "accel",
          "desc": "Accelerator control",
          "max": 225,
          "min": 180
```

(continues on next page)

(continued from previous page)


```
        },
        {
          "name": "node",
          "desc": "Node manager control",
          "max": 410,
          "min": 275
        }
      ]
    },
    {
      "name": "01:000d:206d:0104:0010:0040:3200:0000",
      "desc": "ComputeANC_SNB_260W_16c_64GB_12800_NoAccel",
      "host_limit_max": 350,
      "host_limit_min": 200,
      "static": 0,
      "supply": 425,
      "powerup": 150,
      "nids": [
        40,
        41,
        42,
        43
      ],
      "controls": [
        {
          "name": "node",
          "desc": "Node manager control",
          "max": 350,
          "min": 200
        }
      ]
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **groups** (*object[]*) – Object array containing hardware specific information and NID membership, each element represents a unique hardware type
- **groups[].name** (*string*) – Opaque identifier which Cray system management software uses to uniquely identify a node type
- **groups[].desc** (*string*) – Text description of the opaque node type identifier
- **groups[].host_limit_max** (*int*) – Estimated maximum power, specified in watts, which host CPU(s) and memory may consume
- **groups[].host_limit_min** (*int*) – Estimated minimum power, specified in watts, which host CPU(s) and memory require to operate
- **groups[].static** (*int*) – Static per node power overhead, specified in watts, which is unreported

- **groups[].supply** (*int*) – Maximum capacity of each node level power supply for the given hardware type, specified in watts
- **groups[].powerup** (*int*) – Typical power consumption of each node during hardware initialization, specified in watts
- **groups[].nids** (*int[]*) – NID members belonging to the given hardware type
- **groups[].controls** (*object[]*) – Array of node level control objects which may be assigned or queried, one element per control
- **groups[].controls[].name** (*string*) – Unique control object identifier
- **groups[].controls[].desc** (*string*) – Human readable description of the control object
- **groups[].controls[].min** (*int*) – Minimum value which may be assigned to the control object, units are dependent upon control type
- **groups[].controls[].max** (*int*) – Maximum value which may be assigned to the control object, units are dependent upon control type

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

3.2 get_power_cap

The **get_power_cap** call returns the power capping control(s) and currently applied settings for the requested list of NIDs. Control values which are returned as zero have special meaning. In such case, a zero value indicates the respective control is unconstrained.

3.2.1 CLI Interface

capmc get_power_cap

-n, --nids <nidlist>
Specify the NIDs to query node level, and if applicable, accelerator level power caps. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs. The specified NIDs must be in the **ready** state per the **node_status** command.

```
$ capmc get_power_cap --nids=40,48
{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "nid": 40,
      "controls": [
...
```

3.2.2 HTTP Interface

POST /capmc/get_power_cap
Example Request:

```
POST /capmc/get_power_cap HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 35
Content-Type: application/json

{
  "nids": [
    40,
    48,
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "nid": 40,
      "controls": [
        {
          "name": "node",
          "val": 350
        },
        {
          "name": "node-biased",
          "val": 355
        },
        {
          "name": "bias-factor",
          "val": 1.015471
        }
      ]
    },
    {
      "nid": 48,
      "controls": [
        {
          "name": "node",
          "val": 0
        },
        {
          "name": "accel",
          "val": 0
```

(continues on next page)

(continued from previous page)

```
        }
      ]
    }
  ]
}
```

Example Response: (Partial Success or Failure)

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 52,
  "err_msg": "Invalid exchange",
  "nids": [
    {
      "nid": 40,
      "e": 52,
      "err_msg": "Invalid exchange"
    },
    {
      "nid": 48,
      "controls": [
        {
          "name": "node",
          "val": 0
        },
        {
          "name": "accel",
          "val": 0
        }
      ]
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Overall request status code, zero on total success, non-zero if one or more node specific operations fail
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned control objects
- **nids[].e** (*int*) – Optional, error status, non-zero indicates operation failed on this node
- **nids[].err_msg** (*string*) – Optional, human readable error message applicable to this node
- **nids[].controls** (*object[]*) – Optional, array of node level control and status objects which have been queried, one element per control
- **nids[].controls[].name** (*string*) – Unique control or status object identifier
- **nids[].controls[].val** (*int*) – Control object setting, or zero to indicate control is unconstrained, units are dependent upon control type

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

3.3 set_power_cap

The **set_power_cap** call is used to establish an upper bound with respect to power consumption on a per-node, and if applicable, a sub-node basis. Established power cap parameters will revert to the default configuration on the next system boot.

If setting multiple different power caps is desired, then it is recommended that those be set programmatically via the **json** command with an input data structure conforming to the HTTP Interface request format for this command. The **json** command allows third party software to pass its own JSON formatted requests in a single transaction to the HTTP API service.

Note: Service nodes may not be power capped. If service node NIDs are specified then the request will fail with an invalid parameters error. When applying a power cap, unspecified controls are reset to their default value.

3.3.1 CLI Interface

capmc set_power_cap

-n, --nids <nidlist>
Specify the NIDs to apply the specified power caps. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). This option may not be omitted. The specified NIDs must be in the **ready** state per the **node_status** command.

-N, --node <watts>
Specify the desired node level power cap. The value given must be within the range returned in the capabilities output. A value of zero may be supplied to explicitly clear an existing node level power cap.

Nodes with high powered accelerators and high TDP processors will be automatically power capped at the "supply" limit returned per the **get_power_cap_capabilities** command. If a node level power cap is specified that is within the node control range but exceeds the supply limit, the actual power cap assigned will be clamped at the supply limit.

-A, --accel <watts>
Specify the desired accelerator component power cap. The value given must be within the range returned in the capabilities output. A value of zero may be supplied to to explicitly clear an accelerator power cap.

The accelerator power cap value represents a subset of the total node level power cap. If a node level power cap of 400 watts is applied and an accelerator power cap of 180 watts is applied, then the total node power consumption is limited to 400 watts. If the accelerator is actively consuming its entire 180 watt power allocation, then the host processor, memory subsystem, and support logic for that node may consume a maximum of 220 watts.

3.3.2 HTTP Interface

POST /capmc/set_power_cap
Example Request:

```
POST /capmc/set_power_cap HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 472
Content-Type: application/json

{
  "nids": [
    {
      "nid": 20,
      "controls": [
        {
          "name": "node",
          "val": 400
        }
      ]
    },
    {
      "nid": 21,
      "controls": [
        {
          "name": "node",
          "val": 400
        }
      ]
    },
    {
      "nid": 60,
      "controls": [
        {
          "name": "node",
          "val": 410
        },
        {
          "name": "accel",
          "val": 220
        }
      ]
    }
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes an array of objects which identify target component NIDs, control names, and their associated set point values.

JSON Parameters

- **nids** (*object[]*) – Object array containing NID specific input data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the specified input control objects
- **nids[].controls** (*object[]*) – Array of node level control objects to be adjusted, one element per control
- **nids[].controls[].name** (*string*) – Unique control object identifier
- **nids[].controls[].val** (*int*) – Control object setting, or zero to indicate control is unconstrained, units are dependent upon control type

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "e":0,
    "err_msg":""
}
```

Example Response: (Partial Success or Failure)

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 52,
  "err_msg": "Invalid exchange",
  "nids": [
    {
      "nid": 60,
      "e": 52,
      "err_msg": "Invalid exchange"
    }
  ]
}
```

In the common case, the response payload is short and consists only of an integer status code and an optional message. However there may be instances, likely due to hardware errors, where a small number of nodes encounter a problem and are unable to comply with the command. If an error does occur, extra information pertaining to the specific component where the error occurred is included in the response payload.

JSON Parameters

- **e** (*int*) – Request status code, zero on success.
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific error data, NIDs which experienced success are omitted
- **nids[].nid** (*int*) – NID number owning the returned error data
- **nids[].e** (*int*) – Error status, non-zero indicates operation failed on this node
- **nids[].err_msg** (*string*) – Human readable error string applicable to this node

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

3.4 get_power_bias

The **get_power_bias** API call informs third-party software what, if any, per node multiplication factor will be considered by low level HSS software when applying a node level power cap. When low level HSS assigns a node level power cap, it assigns the product of the caller specified value and the per node bias factor as the actual power cap. By default, each node is assigned a power bias factor of 1.0. This results in the actual power cap being equal to the caller specified value unless a power bias factor has been explicitly configured.

3.4.1 HTTP Interface

POST /capmc/get_power_bias

Example Request:

```
POST /capmc/get_power_bias HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 35
Content-Type: application/json

{
  "nids": [
    40,
    41
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success",
  "nids": [
    {
      "nid": 41,
      "power-bias": 1.0
    },
    {
      "nid": 40,
      "power-bias": 1.0
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Overall request status code, zero on total success, non-zero if one or more node specific operations fail
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned power-bias value
- **nids[].power-bias** (*float*) – Power bias setting, or 1.0 to indicate the default

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

3.5 set_power_bias

A caller may establish a per node power capping bias factor via the **set_power_bias** API call. This may be used as a fine grained tuning knob intended to equalize node to node performance variation, through the dithering of individual node level power caps, while operating the system under a global power cap. A caller may derive the power cap bias factors by any means, or use the built in **set_power_bias_data** and **compute_power_bias** API calls.

Note: Newly established power cap bias factors do not take effect until the respective node level power cap has been reapplied.

3.5.1 HTTP Interface

POST /capmc/set_power_bias

Example Request:

```
POST /capmc/set_power_bias HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 139
Content-Type: application/json

{
  "nids": [
    {
      "nid": 41,
      "power-bias": 0.984529
    },
    {
      "nid": 42,
      "power-bias": 1.015471
    }
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a list of objects where each element contains two values which identify a specific NID and an associated power bias value.

JSON Parameters

- **nids** (*object[]*) – Object array containing NID specific input data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the assigned power-bias value
- **nids[].power-bias** (*float*) – Power bias setting, or 1.0 to reset to default

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

3.6 clr_power_bias

A caller may clear per node power capping bias factors via the **clr_power_bias** API call. This call differs from calling **set_power_bias** with a specified power bias of 1.0 in that this call results in all internal records relating to the assigned power bias being deleted.

3.6.1 HTTP Interface

POST /capmc/clr_power_bias

Example Request:

```
POST /capmc/clr_power_bias HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length:35
Content-Type: application/json

{
  "nids": [
    41,
    42
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

3.7 set_power_bias_data

Average power per NID over an application run may be stored for later processing using the **set_power_bias_data** API call. This information is primarily used within the **compute_power_bias** API call. It is intended that this call be used as part of a higher level system characterization process.

3.7.1 HTTP Interface

POST /capmc/set_power_bias_data

Example Request:

```
POST /capmc/set_power_bias_data HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 141
Content-Type: application/json

{
  "app": "stress",
  "nids": [
    {
      "avgpwr": 350,
      "nid": 41
    },
    {
      "avgpwr": 361,
      "nid": 42
    }
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes an application name and list of objects where each element contains two values which identify a specific NID and an associated average power value.

JSON Parameters

- **nids** (*object[]*) – Object array containing NID specific input data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the assigned power-bias value
- **nids[].avgpwr** (*int*) – Average power consumption over the application run, specified in watts

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

3.8 compute_power_bias

The **compute_power_bias** API call is used to calculate a per node power cap multiplication factor for each NID as it relates to a larger set of NIDs for a given application. The computed values returned by this API call are not automatically saved. If so desired, the values must be explicitly saved using the **set_power_bias** API call. Prior to using this API call, the caller must have previously stored average power data for the specified application and target NID list using the **set_power_bias_data** API call.

3.8.1 HTTP Interface

POST /capmc/compute_power_bias

Example Request:

```
POST /capmc/compute_power_bias HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 55
Content-Type: application/json

{
  "app": "stress",
  "nids": [
    41,
    42
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes two arguments which identify a NID list and application name.

JSON Parameters

- **app** (*string*) – Application name
- **nids** (*int[]*) – User specified list, or empty array for all NIDs in which an average application power record for the specified application exists

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success",
  "nids": [
    {
      "nid": 41,
      "power-bias": 0.984529
    },
    {
      "nid": 42,
      "power-bias": 1.015471
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Overall request status code, zero on total success, non-zero if one or more node specific operations fail
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned power-bias value
- **nids[].power-bias** (*float*) – Power bias setting

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

CHAPTER

FOUR

NODE FREQUENCY & SLEEP STATE CONTROL

CAPMC node frequency and sleep state controls implement an interface such that external agents may modify, on the fly, CPU operating frequencies and sleep state limits. For example, an external agent may reallocate power amongst nodes by adjusting operating frequencies. A system administrator may disable package sleep states on idle nodes in an effort to keep system power consumption inside the power band negotiated with the utility provider. Or, batch schedulers may optimize power efficiency by running phases of an application at differing frequencies. The controls are provided as a means for third-party integrators to implement advanced power management policies.

Runtime P-State and C-State controls are new in SMW-8.0 / CLE-6.0.

4.1 get_freq_capabilities

The **get_freq_capabilities** call informs third-party software about supported processor operating frequencies. Valid operating frequencies, specified in Hz, are returned in list form.

4.1.1 CLI Interface

capmc get_freq_capabilities

-n, --nids <nidlist>
: Specify the NIDs for which to query allowable frequency limits. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). This is a required option.

```
$ capmc get_freq_capabilities --nids=50
{
    "e": 0,
    "err_msg": "",
    "nids": [
      {
        "data": {
          "PWR_Attrs": [
            {
              "PWR_AttrName": "PWR_ATTR_FREQ",
...
```

4.1.2 HTTP Interface

POST /capmc/cnctl
: **Example request**:

```
POST /capmc/cnctl HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 234
Content-Type: application/json

{
  "nids": [
    50
  ],
  "data": {
    "PWR_Function": "PWR_ObjAttrValueCapabilities",
    "PWR_Attrs": [
      {
        "PWR_AttrName": "PWR_ATTR_FREQ"
      }
    ],
    "PWR_MajorVersion": 0,
    "PWR_MinorVersion": 1
  }
}
```

The request must **POST** a JSON object to the API server containing a remote API call function name, a single element list containing the appropriate attribute name, and the protocol version.

JSON Parameters

- **nids** (*int[]*) – User specified NID list, must **not** be empty
- **data** (*object*) – Remote API call input object
- **data.PWR_Function** (*string*) – Remote API call function name, the caller must specify "PWR_ObjAttrValueCapabilities"
- **data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name, the caller must specify "PWR_ATTR_FREQ"
- **data.PWR_MajorVersion** (*int*) – Remote API call major version number, the caller must specify "0"
- **data.PWR_MinorVersion** (*int*) – Remote API call minor version number, the caller must specify "1"

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success",
  "nids": [
    {
      "nid": 50,
      "data": {
        "PWR_Attrs": [
          {
            "PWR_AttrName": "PWR_ATTR_FREQ",
```

(continues on next page)

(continued from previous page)


```
            "PWR_AttrValueCapabilities": [
              2601000,
              2600000,
              2500000,
              2400000,
              2300000,
              2200000,
              2100000,
              2000000,
              1900000,
              1800000,
              1700000,
              1600000,
              1500000,
              1400000,
              1300000,
              1200000
            ],
            "PWR_ReturnCode": 0
          }
        ],
        "PWR_ReturnCode": 0,
        "PWR_ErrorMessages": null,
        "PWR_Messages": null,
        "PWR_MajorVersion": 0,
        "PWR_MinorVersion": 1
      }
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Overall request status code, non-zero on partial success or failure
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned attribute objects
- **nids[].data** (*object*) – Remote API call result object
- **nids[].data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **nids[].data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name copied from original request
- **nids[].data.PWR_Attrs[].PWR_AttrValueCapabilities** (*int[]*) – List of supported processor operating frequencies, in Hz
- **nids[].data.PWR_Attrs[].PWR_ReturnCode** (*int*) – Attribute probe specific result code, non-zero on failure
- **nids[].data.PWR_ReturnCode** (*int*) – Per-node attribute probe result code, non-zero on failure
- **nids[].data.PWR_ErrorMessages** (*string*) – Per-node attribute error message, or null

- **nids[].data.PWR_Messages** (*string*) – Per-node attribute info message, or null
- **nids[].data.PWR_MajorVersion** (*int*) – Remote API call major version number
- **nids[].data.PWR_MinorVersion** (*int*) – Remote API call minor version number

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad Request
- 500 Internal Server Error – Internal command failure
- 504 Gateway Timeout – Gateway Timeout

4.2 get_freq_limits

The **get_freq_limits** call returns the minimum and maximum allowable operating frequencies on a per-node basis. The processor frequency operating window may be constrained from defaults using the **set_freq_limits** call.

4.2.1 CLI Interface

capmc get_freq_limits

-n, --nids <nidlist>
Specify the NIDs for which to query currently active frequency limits. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). This is a required option.

```
$ capmc get_freq_limits --nids=50
{
    "e": 0,
    "err_msg": "",
...
```

4.2.2 HTTP Interface

POST /capmc/cnctl
Example request:

```
POST /capmc/cnctl HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 234
Content-Type: application/json

{
  "nids": [
    50
  ],
  "data": {
    "PWR_Function": "PWR_ObjAttrGetValues",
    "PWR_Attrs": [
      {
        "PWR_AttrName": "PWR_ATTR_FREQ"
```

(continues on next page)

(continued from previous page)

```
        },
        {
          "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MIN"
        },
        {
          "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MAX"
        }
      ],
      "PWR_MajorVersion": 0,
      "PWR_MinorVersion": 1
    }
}
```

The request must **POST** a JSON object to the API server containing a remote API call function name, a three element list containing the appropriate attribute names, and the protocol version.

JSON Parameters

- **nids** (*int[]*) – User specified NID list, must **not** be empty
- **data** (*object*) – Remote API call input object
- **data.PWR_Function** (*string*) – Remote API call function name, the caller must specify "PWR_ObjAttrGetValues"
- **data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name, the caller must specify array elements for attributes named "PWR_ATTR_FREQ", "PWR_ATTR_FREQ_LIMIT_MIN", and "PWR_ATTR_FREQ_LIMIT_MAX"
- **data.PWR_MajorVersion** (*int*) – Remote API call major version number, the caller must specify "0"
- **data.PWR_MinorVersion** (*int*) – Remote API call minor version number, the caller must specify "1"

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success",
  "nids": [
    {
      "nid": 50,
      "data": {
        "PWR_Attrs": [
          {
            "PWR_AttrName": "PWR_ATTR_FREQ",
            "PWR_AttrValue": 2601000,
            "PWR_ReturnCode": 0,
            "PWR_TimeNanoseconds": 80864412,
            "PWR_TimeSeconds": 1433536488
          },
          {
            "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MAX",
```

(continues on next page)

(continued from previous page)


```
            "PWR_AttrValue": 2601000,
            "PWR_ReturnCode": 0,
            "PWR_TimeNanoseconds": 80876654,
            "PWR_TimeSeconds": 1433536488
          },
          {
            "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MIN",
            "PWR_AttrValue": 1200000,
            "PWR_ReturnCode": 0,
            "PWR_TimeNanoseconds": 80883584,
            "PWR_TimeSeconds": 1433536488
          }
        ],
        "PWR_ErrorMessages": null,
        "PWR_MajorVersion": 0,
        "PWR_Messages": null,
        "PWR_MinorVersion": 1,
        "PWR_ReturnCode": 0
      }
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Overall request status code, non-zero on partial success or failure
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned attribute objects
- **nids[].data** (*object*) – Remote API call result object
- **nids[].data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **nids[].data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name copied from original request
- **nids[].data.PWR_Attrs[].PWR_AttrValue** (*int[]*) – Returned attribute value, in Hz
- **nids[].data.PWR_Attrs[].PWR_ReturnCode** (*int*) – Attribute probe specific result code, non-zero on failure
- **nids[].data.PWR_Attrs[].PWR_TimeSeconds** (*int*) – Elapsed seconds since the epoch, in UTC
- **nids[].data.PWR_Attrs[].PWR_TimeNanoseconds** (*int*) – Nanosecond timestamp component
- **nids[].data.PWR_ReturnCode** (*int*) – Per-node attribute probe result code, non-zero on failure
- **nids[].data.PWR_ErrorMessages** (*string*) – Per-node attribute error message, or null
- **nids[].data.PWR_Messages** (*string*) – Per-node attribute info message, or null

- **nids[].data.PWR_MajorVersion** (*int*) – Remote API call major version number
- **nids[].data.PWR_MinorVersion** (*int*) – Remote API call minor version number

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad Request
- 500 Internal Server Error – Internal command failure
- 504 Gateway Timeout – Gateway Timeout

4.3 set_freq_limits

The **set_freq_limits** call allows third-party software to constrain the range of frequencies a node's processor(s) may operate at. Valid values for minimum and maximum frequency are returned via the **get_freq_capabilities** call.

Note: This call may interact with the **aprun –p-state** switch. The ALPS command switch instructs the host processor to run at a fixed frequency for the duration of the application run. If the user requested performance state is not within the range specified in the **set_freq_limits** call, the actual performance state will be capped or floored such that it remains within the specified range.

4.3.1 CLI Interface

capmc set_freq_limits

-n, --nids <nidlist>
Specify the NIDs for which to set frequency limits. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). This is a required option.

-m, --min <min frequency>
Specify the minimum operating frequency. This is a required option.

-M, --max <max frequency>
Specify the maximum operating frequency. This is a required option.

```
$ capmc set_freq_limits --nids=50 --min=1400000 --max=2500000
{
    "e": 0,
    "err_msg": "",
...
```

4.3.2 HTTP Interface

POST /capmc/cnctl
Example request:

```
POST /capmc/cnctl HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 484
```

(continues on next page)

(continued from previous page)

```
Content-Type: application/json

{
  "nids": [
    50
  ],
  "data": {
    "PWR_Function": "PWR_ObjAttrSetValues",
    "PWR_Attrs": [
      {
        "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MIN",
        "PWR_AttrValue": "1400000"
      },
      {
        "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MAX",
        "PWR_AttrValue": "2500000"
      },
      {
        "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MIN",
        "PWR_AttrValue": "1400000"
      }
    ],
    "PWR_MajorVersion": 0,
    "PWR_MinorVersion": 1
  }
}
```

The request must **POST** a JSON object to the API server containing a remote API call function name, a list containing the appropriate attribute names, requested frequency limit values, and the protocol version. Due to internal bounds checking within the processor, it **is** necessary to specify the minimum, maximum, and a repeated minimum attribute value.

JSON Parameters

- **nids** (*int[]*) – User specified NID list, must **not** be empty
- **data** (*object*) – Remote API call input object
- **data.PWR_Function** (*string*) – Remote API call function name, the caller must specify "PWR_ObjAttrValueCapabilities"
- **data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name, the caller must specify array elements for attributes named "PWR_ATTR_FREQ_LIMIT_MIN" and "PWR_ATTR_FREQ_LIMIT_MAX"
- **data.PWR_Attrs[].PWR_AttrValue** (*string*) – Attribute value
- **data.PWR_MajorVersion** (*int*) – Remote API call major version number, the caller must specify "0"
- **data.PWR_MinorVersion** (*int*) – Remote API call minor version number, the caller must specify "1"

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)


```
{
  "e": 0,
  "err_msg": "Success",
  "nids": [
    {
      "nid": 50,
      "data": {
        "PWR_Attrs": [
          {
            "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MIN",
            "PWR_ReturnCode": 0
          },
          {
            "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MAX",
            "PWR_ReturnCode": 0
          },
          {
            "PWR_AttrName": "PWR_ATTR_FREQ_LIMIT_MIN",
            "PWR_ReturnCode": 0
          }
        ],
        "PWR_ErrorMessages": null,
        "PWR_MajorVersion": 0,
        "PWR_Messages": null,
        "PWR_MinorVersion": 1,
        "PWR_ReturnCode": 0
      }
    }
  ]
}
```


The result status may indicate a non-zero return code for the first minimum frequency limit setting. This can happen, if for example, the previous maximum limit setting is less than the newly requested minimum. The operation may be considered successful as long as the maximum and second minimum frequency limit setting result is zero.

JSON Parameters

- **e** (*int*) – Overall request status code, non-zero on partial success or failure
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned attribute objects
- **nids[].data** (*object*) – Remote API call result object
- **nids[].data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **nids[].data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name copied from original request
- **nids[].data.PWR_Attrs[].PWR_ReturnCode** (*int*) – Attribute set specific result code, non-zero on failure
- **nids[].data.PWR_ReturnCode** (*int*) – Per-node attribute set result code, non-zero on failure

- **nids[].data.PWR_ErrorMessages** (*string*) – Per-node attribute error message, or null
- **nids[].data.PWR_Messages** (*string*) – Per-node attribute info message, or null
- **nids[].data.PWR_MajorVersion** (*int*) – Remote API call major version number
- **nids[].data.PWR_MinorVersion** (*int*) – Remote API call minor version number

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad Request
- 500 Internal Server Error – Internal command failure
- 504 Gateway Timeout – Gateway Timeout

4.4 get_sleep_state_limit_capabilities

The **get_sleep_state_limit_capabilities** call informs third-party software about supported processor sleep states. Valid sleep states are returned in list form. Higher sleep state numbers correspond to deeper sleep states.

4.4.1 CLI Interface

capmc get_sleep_state_limit_capabilities

-n, --nids <nidlist>
Specify the NIDs for which to query allowable sleep state limits. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). This is a required option.

```
$ capmc get_sleep_state_limit_capabilities --nids=50
{
    "e": 0,
    "err_msg": "",
...
```

4.4.2 HTTP Interface

POST /capmc/cnctl
Example request:

```
POST /capmc/cnctl HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 242
Content-Type: application/json

{
  "nids": [
    50
  ],
  "data": {
    "PWR_Function": "PWR_ObjAttrValueCapabilities",
    "PWR_Attrs": [
```

(continues on next page)

(continued from previous page)

```
      {
        "PWR_AttrName": "PWR_ATTR_CSTATE_LIMIT"
      }
    ],
    "PWR_MajorVersion": 0,
    "PWR_MinorVersion": 1
  }
}
```

The request must **POST** a JSON object to the API server containing a remote API call function name, a single element list containing the appropriate attribute name, and the protocol version.

JSON Parameters

- **nids** (*int[]*) – User specified NID list, must **not** be empty
- **data** (*object*) – Remote API call input object
- **data.PWR_Function** (*string*) – Remote API call function name, the caller must specify "PWR_ObjAttrValueCapabilities"
- **data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name, the caller must specify "PWR_ATTR_CSTATE_LIMIT"
- **data.PWR_MajorVersion** (*int*) – Remote API call major version number, the caller must specify "0"
- **data.PWR_MinorVersion** (*int*) – Remote API call minor version number, the caller must specify "1"

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success",
  "nids": [
    {
      "nid": 50,
      "data": {
        "PWR_Attrs": [
          {
            "PWR_AttrName": "PWR_ATTR_CSTATE_LIMIT",
            "PWR_AttrValueCapabilities": [
              0,
              1,
              2,
              3,
              4,
              5
            ],
            "PWR_ReturnCode": 0
          }
        ],
        "PWR_ErrorMessages": null,
```

(continues on next page)

(continued from previous page)


```
            "PWR_MajorVersion": 0,
            "PWR_Messages": null,
            "PWR_MinorVersion": 1,
            "PWR_ReturnCode": 0
          }
        }
      ]
    }
```


JSON Parameters

- **e** (*int*) – Overall request status code, non-zero on partial success or failure
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned attribute objects
- **nids[].data** (*object*) – Remote API call result object
- **nids[].data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **nids[].data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name copied from original request
- **nids[].data.PWR_Attrs[].PWR_AttrValueCapabilities** (*int[]*) – List of supported sleep states
- **nids[].data.PWR_Attrs[].PWR_ReturnCode** (*int*) – Attribute probe specific result code, non-zero on failure
- **nids[].data.PWR_ReturnCode** (*int*) – Per-node attribute probe result code, non-zero on failure
- **nids[].data.PWR_ErrorMessages** (*string*) – Per-node attribute error message, or null
- **nids[].data.PWR_Messages** (*string*) – Per-node attribute info message, or null
- **nids[].data.PWR_MajorVersion** (*int*) – Remote API call major version number
- **nids[].data.PWR_MinorVersion** (*int*) – Remote API call minor version number

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad Request
- 500 Internal Server Error – Internal command failure
- 504 Gateway Timeout – Gateway Timeout

4.5 get_sleep_state_limit

The **get_sleep_state_limit** call returns the state number identifying the deepest allowable sleep on a per-node basis. The deepest allowable sleep state limit may be constrained from defaults using the **set_sleep_state_limit** call.

4.5.1 CLI Interface

capmc get_sleep_state_limit

-n, --nids <nidlist>
Specify the NIDs for which to query the currently active state limit. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). This is a required option.

```
$ capmc get_sleep_state_limit --nids=50
{
    "e": 0,
    "err_msg": "",
...
```

4.5.2 HTTP Interface

POST /capmc/cnctl
Example request:

```
POST /capmc/cnctl HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 234
Content-Type: application/json

{
  "nids": [
    50
  ],
  "data": {
    "PWR_Function": "PWR_ObjAttrGetValues",
    "PWR_Attrs": [
      {
        "PWR_AttrName": "PWR_ATTR_CSTATE_LIMIT"
      }
    ],
    "PWR_MajorVersion": 0,
    "PWR_MinorVersion": 1
  }
}
```

The request must **POST** a JSON object to the API server containing a remote API call function name, a single element list containing the appropriate attribute name, and the protocol version.

JSON Parameters

- **nids** (*int[]*) – User specified NID list, must **not** be empty
- **data** (*object*) – Remote API call input object
- **data.PWR_Function** (*string*) – Remote API call function name, the caller must specify "PWR_ObjAttrGetValues"
- **data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name, the caller must specify "PWR_ATTR_CSTATE_LIMIT"

- **data.PWR_MajorVersion** (*int*) – Remote API call major version number, the caller must specify "0"
- **data.PWR_MinorVersion** (*int*) – Remote API call minor version number, the caller must specify "1"

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success",
  "nids": [
    {
      "nid": 50,
      "data": {
        "PWR_Attrs": [
          {
            "PWR_AttrName": "PWR_ATTR_CSTATE_LIMIT",
            "PWR_AttrValue": 5,
            "PWR_ReturnCode": 0,
            "PWR_TimeNanoseconds": 777098381,
            "PWR_TimeSeconds": 1433536488
          }
        ],
        "PWR_ErrorMessages": null,
        "PWR_MajorVersion": 0,
        "PWR_Messages": null,
        "PWR_MinorVersion": 1,
        "PWR_ReturnCode": 0
      }
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Overall request status code, non-zero on partial success or failure
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned attribute objects
- **nids[].data** (*object*) – Remote API call result object
- **nids[].data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **nids[].data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name copied from original request
- **nids[].data.PWR_Attrs[].PWR_AttrValue** (*int[]*) – Returned attribute value
- **nids[].data.PWR_Attrs[].PWR_ReturnCode** (*int*) – Attribute probe specific result code, non-zero on failure
- **nids[].data.PWR_Attrs[].PWR_TimeSeconds** (*int*) – Elapsed seconds since the epoch, in UTC

- **nids[].data.PWR_Attrs[].PWR_TimeNanoseconds** (*int*) – Nanosecond timestamp component
- **nids[].data.PWR_ReturnCode** (*int*) – Per-node attribute probe result code, non-zero on failure
- **nids[].data.PWR_ErrorMessages** (*string*) – Per-node attribute error message, or null
- **nids[].data.PWR_Messages** (*string*) – Per-node attribute info message, or null
- **nids[].data.PWR_MajorVersion** (*int*) – Remote API call major version number
- **nids[].data.PWR_MinorVersion** (*int*) – Remote API call minor version number

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad Request
- 500 Internal Server Error – Internal command failure
- 504 Gateway Timeout – Gateway Timeout

4.6 set_sleep_state_limit

The **set_sleep_state_limit** call allows third-party software to constrain the deepest sleep state a node's processor(s) may enter. Valid values for sleep state limits are returned via the **get_sleep_state_limit_capabilities** call.

Note: Disabling sleep states, in some circumstances, can result in a slight loss of performance. This is due in part because idle hardware components which may have otherwise entered a low power state are instead forced to busy wait. This may cause resource contention and consume excessive power, subtracting from the available resources of those components performing useful work.

4.6.1 CLI Interface

capmc set_sleep_state_limit

-n, --nids <nidlist>
Specify the NIDs for which to set the currently active state limit. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). This is a required option.

-l, --limit <sleep state limit>
Specify the sleep state limit to put in place on the target NIDs. Valid limits are returned via the **get_sleep_state_limit_capabilities** call.

```
$ capmc set_sleep_state_limit --nids=50 --limit=4
{
    "e": 0,
    "err_msg": "",
...
```

4.6.2 HTTP Interface

POST /capmc/cnctl

Example request:

```
POST /capmc/cnctl HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 265
Content-Type: application/json

{
  "nids": [
    50
  ],
  "data": {
    "PWR_Function": "PWR_ObjAttrSetValues",
    "PWR_Attrs": [
      {
        "PWR_AttrName": "PWR_ATTR_CSTATE_LIMIT",
        "PWR_AttrValue": "4"
      }
    ],
    "PWR_MajorVersion": 0,
    "PWR_MinorVersion": 1
  }
}
```

The request must **POST** a JSON object to the API server containing a remote API call function name, a single element list containing the appropriate attribute name, requested sleep state limit value, and the protocol version.

JSON Parameters

- **nids** (*int[]*) – User specified NID list, must **not** be empty
- **data** (*object*) – Remote API call input object
- **data.PWR_Function** (*string*) – Remote API call function name, the caller must specify "PWR_ObjAttrValueCapabilities"
- **data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name, the caller must specify "PWR_ATTR_CSTATE_LIMIT"
- **data.PWR_Attrs[].PWR_AttrValue** (*string*) – Attribute value
- **data.PWR_MajorVersion** (*int*) – Remote API call major version number, the caller must specify "0"
- **data.PWR_MinorVersion** (*int*) – Remote API call minor version number, the caller must specify "1"

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success",
```

(continues on next page)

(continued from previous page)


```
  "nids": [
    {
      "nid": 50,
      "data": {
        "PWR_Attrs": [
          {
            "PWR_AttrName": "PWR_ATTR_CSTATE_LIMIT",
            "PWR_ReturnCode": 0
          }
        ],
        "PWR_ErrorMessages": null,
        "PWR_MajorVersion": 0,
        "PWR_Messages": null,
        "PWR_MinorVersion": 1,
        "PWR_ReturnCode": 0
      }
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Overall request status code, non-zero on partial success or failure
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array containing NID specific result data, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned attribute objects
- **nids[].data** (*object*) – Remote API call result object
- **nids[].data.PWR_Attrs** (*object[]*) – Array of attribute name value pairs
- **nids[].data.PWR_Attrs[].PWR_AttrName** (*string*) – Attribute name copied from original request
- **nids[].data.PWR_Attrs[].PWR_ReturnCode** (*int*) – Attribute set specific result code, non-zero on failure
- **nids[].data.PWR_ReturnCode** (*int*) – Per-node attribute set result code, non-zero on failure
- **nids[].data.PWR_ErrorMessages** (*string*) – Per-node attribute error message, or null
- **nids[].data.PWR_Messages** (*string*) – Per-node attribute info message, or null
- **nids[].data.PWR_MajorVersion** (*int*) – Remote API call major version number
- **nids[].data.PWR_MinorVersion** (*int*) – Remote API call minor version number

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad Request
- 500 Internal Server Error – Internal command failure
- 504 Gateway Timeout – Gateway Timeout

CHAPTER

FIVE

NODE MCDRAM CAPABILITIES & CONTROL

CAPMC MCDRAM controls implement a simple interface for querying and manipulating Intel© Xeon Phi™ processor x200 node MCDRAM capabilities and configuration. These following API calls enable external software to configure nodes to meet various job or site configuration requirements.

All MCDRAM API calls, except for **get_mcdram_capabilities**, gather or control staged configuration values. Staged values are applied at next boot.

5.1 get_mcdram_capabilities

The **get_mcdram_capabilities** call informs third-party software about the MCDRAM capabilities of Intel Xeon Phi processor nodes. Information returned includes a list objects where each object contains a Intel Xeon Phi processor NID number and a comma-separated list of MCDRAM configuration values that NID supports. Each string configuration value has an equivalent integer representation - both values represent the percentage of MCDRAM to be used as cache. Information may be returned for a targeted set of Intel Xeon Phi processor NIDs or all Intel Xeon Phi processor NIDs in the system. Non-Intel Xeon Phi processor NID targets will be filtered out of a request. If an invalid or undiscovered NID is specified in the target list, an error response will be returned to the caller. No such error will occur, however, if the request's NIDs list is empty.

5.1.1 CLI Interface

capmc get_mcdram_capabilities

-n, --nids <nidlist>
: Specify the NIDs for which to query possible MCDRAM parameters. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

```
$ capmc get_mcdram_capabilities  -n 40-43,48,49
{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "e": 0,
      "err_msg": "",
      "mcdram_cfg": "flat,0,split,25,equal,50,cache,100",
      "nid": 40
    },
    ...
```

5.1.2 HTTP Interface

POST /capmc/get_mcdram_capabilities

Example Request:

```
POST /capmc/get_mcdram_capabilities HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "e": 0,
      "err_msg": "",
      "mcdram_cfg": "flat,0,split,25,equal,50,cache,100",
      "nid": 40
    },
    {
      "e": 0,
      "err_msg": "",
      "mcdram_cfg": "flat,0,split,25,equal,50,cache,100",
      "nid": 41
    },
    {
      "e": 0,
      "err_msg": "",
      "mcdram_cfg": "flat,0,split,25,equal,50,cache,100",
      "nid": 42
    },
```

(continues on next page)

(continued from previous page)


```
    {
      "e": 0,
      "err_msg": "",
      "mcdram_cfg": "flat,0,split,25,equal,50,cache,100",
      "nid": 43
    },
    {
      "e": 0,
      "err_msg": "",
      "mcdram_cfg": "flat,0,split,25,equal,50,cache,100",
      "nid": 48
    },
    {
      "e": 0,
      "err_msg": "",
      "mcdram_cfg": "flat,0,split,25,equal,50,cache,100",
      "nid": 49
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array where each element represents a unique NID and its MCDRAM capability
- **nids[].nid** (*int*) – NID to which the mcdram_cfg applies
- **nids[].mcdram_cfg** (*string*) – CSV list of supported MCDRAM configuration parameters, empty when NID's status code is non-zero
- **nids[].e** (*int*) – NID status code, non-zero if NID is invalid
- **nids[].err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

5.2 get_mcdram_cfg

The **get_mcdram_cfg** call informs third-party software about the staged MCDRAM configuration of Intel Xeon Phi processor nodes, which is the MCDRAM configuration that will be applied at next boot. Information returned includes a list objects where each object contains a Intel Xeon Phi processor NID number and that NIDs staged MCDRAM configuration and percentage, and DRAM and MCDRAM sizes. Information may be returned for a targeted set of Intel Xeon Phi processor NIDs or all Intel Xeon Phi processor NIDs in the system. Non-Intel Xeon Phi processor NID targets will be filtered out of a request, if specified. If an invalid or undiscovered NID is specified, an error response will be returned to the caller. No such error will occur, however, if the request's NIDs list is empty.

5.2.1 CLI Interface

capmc get_mcdram_cfg

-n, --nids <nidlist>
Specify the NIDs for which to query the staged MCDRAM configuration. Does not return current configuration state, but rather the configuration that will be applied at the next boot. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

```
$ capmc get_mcdram_cfg -n 40-43,48,49
{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "mcdram_cfg": "cache",
      "mcdram_pct": 100,
      "mcdram_size": "16384MB",
      "nid": 40,
      "dram_size": "96GB"
    },
    ...
```

5.2.2 HTTP Interface

POST /capmc/get_mcdram_cfg
Example Request:

```
POST /capmc/get_mcdram_cfg HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "mcdram_cfg": "cache",
      "mcdram_pct": 100,
      "mcdram_size": "16384MB",
      "nid": 40,
      "dram_size": "96GB"
    },
    {
      "mcdram_cfg": "cache",
      "mcdram_pct": 100,
      "mcdram_size": "16384MB",
      "nid": 41,
      "dram_size": "96GB"
    },
    {
      "mcdram_cfg": "cache",
      "mcdram_pct": 100,
      "mcdram_size": "16384MB",
      "nid": 42,
      "dram_size": "96GB"
    },
    {
      "mcdram_cfg": "cache",
      "mcdram_pct": 100,
      "mcdram_size": "16384MB",
      "nid": 43,
      "dram_size": "96GB"
    },
    {
      "mcdram_cfg": "cache",
      "mcdram_pct": 100,
      "mcdram_size": "16384MB",
      "nid": 48,
      "dram_size": "96GB"
    },
    {
      "mcdram_cfg": "cache",
      "mcdram_pct": 100,
      "mcdram_size": "16384MB",
      "nid": 49,
      "dram_size": "96GB"
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array where each element represents a unique NID and its

staged MCDRAM configuration

- **nids[].nid** (*int*) – NID to which the staged mcdram_cfg applies
- **nids[].mcdram_cfg** (*string*) – Staged MCDRAM configuration setting that will be applied at next boot
- **nids[].mcdram_pct** (*int*) – Percentage of MCDRAM that will be used as cache, based on the MCDRAM configuration value
- **nids[].mcdram_size** (*string*) – NID's MCDRAM size with unit of measure
- **nids[].dram_size** (*string*) – NID's DRAM size with unit of measure

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

5.3 set_mcdram_cfg

The **set_mcdram_cfg** call allows third-party software to stage MCDRAM configuration on Intel Xeon Phi processor nodes. A node will apply the MCDRAM configuration at its next boot. Configuration can be targeted at a certain set of Intel Xeon Phi processor nodes or all Intel Xeon Phi processor nodes in a system. Non-Intel Xeon Phi processor NID targets, if specified, will be filtered out of a request. A simple object is returned to the caller indicating whether the operation succeeded or failed. If an invalid or undiscovered NID is specified, a more detailed error response will be returned to the caller.

5.3.1 CLI Interface

capmc set_mcdram_cfg

-n, --nids <nidlist>
Specify NIDs for which to set a staged MCDRAM configuration for application at next boot. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

-m, --mode <flat|0|split|25|equal|50|cache|100|reset>
Specify MCDRAM configuration, which is the percentage of MCDRAM to be used as cache. Valid values include: 'flat' (or 0), 'split' (or 25), 'equal' (or 50), and 'cache' (or 100). The value 'reset' can be used to reset MCDRAM configuration back to the default value, which is 'cache.' This is a required option.

```
$ capmc set_mcdram_cfg -n 40-43,48,49 -m cache
{
  "e": 0,
  "err_msg": "Success"
}
```

5.3.2 HTTP Interface

POST /capmc/set_mcdram_cfg
Example Request:

```
POST /capmc/set_mcdram_cfg HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 377
Content-Type: application/json

{
  "nids": [
    {
      "mcdram_cfg": "cache",
      "nid": 40
    },
    {
      "mcdram_cfg": "cache",
      "nid": 41
    },
    {
      "mcdram_cfg": "cache",
      "nid": 42
    },
    {
      "mcdram_cfg": "cache",
      "nid": 43
    },
    {
      "mcdram_cfg": "cache",
      "nid": 48
    },
    {
      "mcdram_cfg": "cache",
      "nid": 49
    }
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies an object list containing objects objects that have two attributes, NID and MCDRAM configuration.

JSON Parameters

- **nids** (*object[]*) – User specified list of objects, where each object contains a pair of attributes, mcdram_cfg and a NID number. This list must not contain invalid or duplicate NID numbers. The api user should call **get_mcdram_capabilities** prior to calling **set_mcdram_cfg** in order to discover which Intel Xeon Phi processor NIDs are available and what their respective MCDRAM capabilities have. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

5.4 clr_mcdram_cfg

The **clr_mcdram_cfg** call allows third-party software to reset the staged MCDRAM configuration on Intel Xeon Phi processor nodes to the default value of 'cache'. A node will apply the MCDRAM configuration at its next boot. Configuration can be targeted at a certain set of Intel Xeon Phi processor nodes or all Intel Xeon Phi processor nodes in a system. Non-Intel Xeon Phi processor NID targets, if specified, will be filtered out of a request. A simple object is returned to the caller indicating whether the operation succeeded or failed. If an invalid or undiscovered NID is specified, a more detailed error response will be returned to the caller.

5.4.1 CLI Interface

capmc clr_mcdram_cfg

The CLI interface for this API call is available using the **-m reset** option of the **set_mcdram_cfg** API call.

```
$ capmc set_mcdram_cfg -n 40-43,48,49 -m reset
{
  "e": 0,
  "err_msg": "Success"
}
```

5.4.2 HTTP Interface

POST /capmc/clr_mcdram_cfg

Example Request:

```
POST /capmc/clr_mcdram_cfg HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

CHAPTER

SIX

NODE NUMA CAPABILITIES & CONTROL

CAPMC NUMA controls implement a simple interface for querying and manipulating Intel© Xeon Phi™ processor x200 node NUMA capabilities and configuration. These following API calls enable external software to configure nodes to meet various job or site configuration requirements.

All NUMA API calls, except for **get_numa_capabilities**, gather or control staged configuration values. Staged values are applied at next boot.

6.1 get_numa_capabilities

The **get_numa_capabilities** call informs third-party software about the NUMA capabilities of Intel Xeon Phi processor nodes. Information returned includes a list objects where each object contains a Intel Xeon Phi processor NID number and a comma-separated list of NUMA configuration values that NID supports. Information may be returned for a targeted set of Intel Xeon Phi processor NIDs or all Intel Xeon Phi processor NIDs in the system. Non-Intel Xeon Phi processor NID targets will be filtered out of a request. If an invalid or undiscovered NID is specified in the target list, an error response will be returned to the caller. No such error will occur, however, if the request's NIDs list is empty.

6.1.1 CLI Interface

capmc get_numa_capabilities

-n, --nids <nidlist>
: Specify the NIDs for which to query possible NUMA parameters. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

```
$ capmc get_numa_capabilities  -n 40-43,48,49
{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "e": 0,
      "err_msg": "",
      "numa_cfg": "a2a,snc2,snc4,hemi,quad",
      "nid": 40
    },
    ...
```

6.1.2 HTTP Interface

POST /capmc/get_numa_capabilities

Example Request:

```
POST /capmc/get_numa_capabilities HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "e": 0,
      "err_msg": "",
      "numa_cfg": "a2a,snc2,snc4,hemi,quad",
      "nid": 40
    },
    {
      "e": 0,
      "err_msg": "",
      "numa_cfg": "a2a,snc2,snc4,hemi,quad",
      "nid": 41
    },
    {
      "e": 0,
      "err_msg": "",
      "numa_cfg": "a2a,snc2,snc4,hemi,quad",
      "nid": 42
    },
```

(continues on next page)

(continued from previous page)


```
    {
      "e": 0,
      "err_msg": "",
      "numa_cfg": "a2a,snc2,snc4,hemi,quad",
      "nid": 43
    },
    {
      "e": 0,
      "err_msg": "",
      "numa_cfg": "a2a,snc2,snc4,hemi,quad",
      "nid": 48
    },
    {
      "e": 0,
      "err_msg": "",
      "numa_cfg": "a2a,snc2,snc4,hemi,quad",
      "nid": 49
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array where each element represents a unique NID and its NUMA capability
- **nids[].nid** (*int*) – NID to which the numa_cfg applies
- **nids[].numa_cfg** (*string*) – CSV list of supported NUMA configuration parameters, empty when NID's status code is non-zero
- **nids[].e** (*int*) – NID status code, non-zero if NID is invalid
- **nids[].err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

6.2 get_numa_cfg

The **get_numa_cfg** call informs third-party software about the staged NUMA configuration of Intel Xeon Phi processor nodes, which is the NUMA configuration that will be applied at next boot. Information returned includes a list objects where each object contains a pair of attributes, a Intel Xeon Phi processor NID number and that NID's staged NUMA configuration. Information may be returned for a targeted set of Intel Xeon Phi processor NIDs or all Intel Xeon Phi processor NIDs in the system. Non-Intel Xeon Phi processor NID targets will be filtered out of a request, if specified. If an invalid or undiscovered NID is specified, an error response will be returned to the caller. No such error will occur, however, if the request's NIDs list is empty.

6.2.1 CLI Interface

capmc get_numa_cfg

-n, --nids <nidlist>
Specify the NIDs for which to query the staged NUMA configuration. Does not return current configuration state, but rather the configuration that will be applied at the next boot. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

```
$ capmc get_numa_cfg -n 40-43,48,49
{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "numa_cfg": "hemi",
      "nid": 40,
    },
    ...
```

6.2.2 HTTP Interface

POST /capmc/get_numa_cfg
Example Request:

```
POST /capmc/get_numa_cfg HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)


```
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "numa_cfg": "hemi",
      "nid": 40
    },
    {
      "numa_cfg": "hemi",
      "nid": 41
    },
    {
      "numa_cfg": "hemi",
      "nid": 42
    },
    {
      "numa_cfg": "hemi",
      "nid": 43
    },
    {
      "numa_cfg": "hemi",
      "nid": 48
    },
    {
      "numa_cfg": "hemi",
      "nid": 49
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array where each element represents a unique NID and its staged NUMA configuration
- **nids[].nid** (*int*) – NID to which the staged numa_cfg applies
- **nids[].numa_cfg** (*string*) – Staged NUMA configuration setting that will be applied at next boot

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

6.3 set_numa_cfg

The **set_numa_cfg** call allows third-party software to stage NUMA configuration on Intel Xeon Phi processor nodes. A node will apply the NUMA configuration at its next boot. Configuration can be targeted at a certain set of Intel Xeon Phi processor nodes or all Intel Xeon Phi processor nodes in a system. Non-Intel Xeon Phi processor NID targets, if

specified, will be filtered out of a request. A simple object is returned to the caller indicating whether the operation succeeded or failed. If an invalid or undiscovered NID is specified, a more detailed error response will be returned to the caller.

6.3.1 CLI Interface

capmc set_numa_cfg

-n, --nids <nidlist>
Specify NIDs for which to set a staged NUMA configuration for application at next boot. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

-m, --mode <a2a|snc2|snc4|hemi|quad|reset>
Specify NUMA configuration mode. Valid values include: 'a2a,' 'snc2,' 'snc4,' 'hemi,' and 'quad'. The value 'reset' can be used to reset NUMA configuration back to the default value, which is 'a2a.' This is a required option.

```
$ capmc set_numa_cfg -n 40-43,48,49 -m hemi
{
  "e": 0,
  "err_msg": "Success"
}
```

6.3.2 HTTP Interface

POST /capmc/set_numa_cfg
Example Request:

```
POST /capmc/set_numa_cfg HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 359
Content-Type: application/json

{
  "nids": [
    {
      "numa_cfg": "hemi",
      "nid": 40
    },
    {
      "numa_cfg": "hemi",
      "nid": 41
    },
    {
      "numa_cfg": "hemi",
      "nid": 42
    },
    {
      "numa_cfg": "hemi",
      "nid": 43
    },
    {
      "numa_cfg": "hemi",
```

(continues on next page)

(continued from previous page)

```
      "nid": 48
    },
    {
      "numa_cfg": "hemi",
      "nid": 49
    }
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies an object list containing objects that have two attributes, NID and NUMA configuration.

JSON Parameters

- **nids** (*object[]*) – User specified list of objects, where each object contains a pair of attributes, numa_cfg and a NID number. This list must not contain invalid or duplicate NID numbers. The API user should call **get_numa_capabilities** prior to calling **set_numa_cfg** in order to discover which Intel Xeon Phi processor NIDs are available and what their respective NUMA capabilities are. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

6.4 clr_numa_cfg

The **clr_numa_cfg** call allows third-party software to reset the staged NUMA configuration on Intel Xeon Phi processor nodes to the default value of 'a2a'. A node will apply the NUMA configuration at its next boot. Configuration can be targeted at a certain set of Intel Xeon Phi processor nodes or all Intel Xeon Phi processor nodes in a system. Non-Intel Xeon Phi processor NID targets, if specified, will be filtered out of a request. A simple object is returned to the caller indicating whether the operation succeeded or failed. If an invalid or undiscovered NID is specified, a more detailed error response will be returned to the caller.

6.4.1 CLI Interface

capmc clr_numa_cfg

The CLI interface for this API call is available using the **-m reset** option of the **set_numa_cfg** API call.

```
$ capmc set_numa_cfg -n 40-43,48,49 -m reset
{
  "e": 0,
  "err_msg": "Success"
}
```

6.4.2 HTTP Interface

POST /capmc/clr_numa_cfg

Example Request:

```
POST /capmc/clr_numa_cfg HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

CHAPTER

SEVEN

NODE SSD CONTROL & DIAGNOSTICS

CAPMC SSD controls implement a simple interface for querying and manipulating Intel© Xeon Phi™ processor x200 node on-board SSDs. These following API calls enable external software to configure nodes to meet various job or site configuration requirements. These calls are limited to Intel Xeon Phi processor nodes.

All Node SSD Control API calls gather or control staged configuration values. Staged values are applied at next boot.

CAPMC SSD diagnostics implement an interface for querying various inventory and diagnositic information from SSDs on a system. SSD diagnostics can target any node in a system, not just Intel Xeon Phi processor nodes.

7.1 get_ssd_enable

SSDs on Intel Xeon Phi processor nodes can have a state of enabled or disabled. The **get_ssd_enable** call informs third-party software about the staged state of SSDs on Intel Xeon Phi processor nodes, which is the SSD state that will be applied at next boot. Information returned includes a list objects where each object contains a pair of attributes, a Intel Xeon Phi processor NID number and the stated state of that NID's SSD. Information may be returned for a targeted set of Intel Xeon Phi processor NIDs or all Intel Xeon Phi processor NIDs in the system. Non-Intel Xeon Phi processor NID targets will be filtered out of a request, if specified. If an invalid or undiscovered NID is specified, an error response will be returned to the caller. No such error will occur, however, if the request's NIDs list is empty.

7.1.1 CLI Interface

capmc get_ssd_enable

-n, --nids <nidlist>
: Specify the NIDs for which to query the staged SSD enabled state. Does not return current SSD state, but rather the SSD state that will be applied at the next boot. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

```
$ capmc get_ssd_enable -n 40-43,48,49
{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "ssd_enable": 1,
      "nid": 40,
    },
    ...
```

7.1.2 HTTP Interface

POST /capmc/get_ssd_enable

Example Request:

```
POST /capmc/get_ssd_enable HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "ssd_enable": 1,
      "nid": 40
    },
    {
      "ssd_enable": 1,
      "nid": 41
    },
    {
      "ssd_enable": 1,
      "nid": 42
    },
    {
      "ssd_enable": 1,
      "nid": 43
    },
    {
      "ssd_enable": 1,
```

(continues on next page)

(continued from previous page)

```
      "nid": 48
    },
    {
      "ssd_enable": 1,
      "nid": 49
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **nids** (*object[]*) – Object array where each element represents a unique NID and its staged SSD state
- **nids[].nid** (*int*) – NID to which the staged ssd_enable setting applies
- **nids[].ssd_enable** (*int*) – Staged SSD state that will be applied at next boot

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

7.2 set_ssd_enable

The **set_ssd_enable** call allows third-party software to stage SSD state on Intel Xeon Phi processor nodes. A node will apply the SSD state value at its next boot. Configuration can be targeted at a certain set of Intel Xeon Phi processor nodes or all Intel Xeon Phi processor nodes in a system. Non-Intel Xeon Phi processor NID targets, if specified, will be filtered out of a request. A simple object is returned to the caller indicating whether the operation succeeded or failed. If an invalid or undiscovered NID is specified, a more detailed error response will be returned to the caller.

7.2.1 CLI Interface

capmc set_ssd_enable

-n, --nids <nidlist>
: Specify NIDs for which to set a staged SSD state value for application at next boot. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

-m, --mode <enable|disable|reset>
: Specify SSD state mode. Valid values include: 'enable,' 'disable,' and 'reset.' The value 'reset' can be used to reset SSD configuration back to the default state value, which is 'enable.' This is a required option.

```
$ capmc set_set_enable -n 40-43,48,49 -m disable
{
  "e": 0,
  "err_msg": "Success"
}
```

7.2.2 HTTP Interface

POST /capmc/set_ssd_enable

Example Request:

```
POST /capmc/set_ssd_enable HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 341
Content-Type: application/json

{
  "nids": [
    {
      "ssd_enable": 1,
      "nid": 40
    },
    {
      "ssd_enable": 1,
      "nid": 41
    },
    {
      "ssd_enable": 1,
      "nid": 42
    },
    {
      "ssd_enable": 1,
      "nid": 43
    },
    {
      "ssd_enable": 1,
      "nid": 48
    },
    {
      "ssd_enable": 1,
      "nid": 49
    }
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies an object list containing objects that have two attributes, NID and SSD state.

JSON Parameters

- **nids** (*object[]*) – User specified list of objects, where each object contains a pair of attributes, ssd_enable and a NID number. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

7.3 clr_ssd_enable

The **clr_ssd_enable** call allows third-party software to reset the staged SSD state on Intel Xeon Phi processor nodes to the default value of '1' (which is 'enable'). A node will apply the SSD state at its next boot. Configuration can be targeted at a certain set of Intel Xeon Phi processor nodes or all Intel Xeon Phi processor nodes in a system. Non-Intel Xeon Phi processor NID targets, if specified, will be filtered out of a request. A simple object is returned to the caller indicating whether the operation succeeded or failed. If an invalid or undiscovered NID is specified, a more detailed error response will be returned to the caller.

7.3.1 CLI Interface

capmc clr_ssd_enable

The CLI interface for this API call is available using the **-m reset** option of the **set_ssd_enable** API call.

```
$ capmc set_ssd_enable -n 40-43,48,49 -m reset
{
  "e": 0,
  "err_msg": "Success"
}
```

7.3.2 HTTP Interface

POST /capmc/clr_ssd_enable

Example Request:

```
POST /capmc/clr_ssd_enable HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 71
Content-Type: application/json

{
  "nids": [
    40,
    41,
    42,
    43,
    48,
    49
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "Success"
}
```

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

7.4 get_ssds

The **get_ssds** call provides third-party software with SSD inventory information. Information returned includes an object with zero or more node cname attributes that each map to a list of one or more SSD inventory objects. Each inventory object can include the SSD's PCI bus, device and function, model number, serial number, and system and sub-system device and vendor IDs. Information may be returned for a targeted set of NIDs or all NIDs in the system. If the targeted NIDs do not have SSDs attached, the response object will only contain a success or failure code and message for the API call itself. If an invalid NID is specified, an error response will be returned to the caller.

7.4.1 CLI Interface

capmc get_ssds

-n, --nids <nidlist>
Specify the NIDs for which to query SSD inventory information. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs. Cannot be used with -c.

-c, --cname <cnamelist>
Specify a comma-separated list of node cnames for which to query SSD inventory information. If omitted, the default is all nodes. Cannot be used with -n.

```
$ capmc get_ssds -c 'c0-0c1s10n0'
{
  "err_msg": "Success",
  "e": 0,
  "c0-0c1s10n0": [
    {
      "sub_id": "144da801",
      "func": 0,
      "serial_number": "S0ABCDAH000000",
      "device": 0,
      "bus": 2,
      "ssd_id": "144da802",
      "model_number": "SAMSUNG MZVPV128HDGM-00000",
      "nid": 104,
      "size": 128000000000
    }
  ]
}
```

7.4.2 HTTP Interface

POST /capmc/get_ssds
Example Request:

```
POST /capmc/get_ssd_enable HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 27
Content-Type: application/json

{
  "nids": [
    104
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes no arguments (ie, an empty object) or a single argument, which is either a list of NID numbers or a list of node cname strings.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.
- **cname** (*string[]*) – User specified list, or empty array for all nodes. This list must not contain invalid cnames or cnames referencing objects that are not nodes. If invalid cnames are specified then an error will be returned.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)


```
{
  "err_msg": "Success",
  "e": 0,
  "c0-0c1s10n0": [
    {
      "sub_id": "144da801",
      "func": 0,
      "serial_number": "S0ABCDAH000000",
      "device": 0,
      "bus": 2,
      "ssd_id": "144da802",
      "model_number": "SAMSUNG MZVPV128HDGM-00000",
      "nid": 104,
      "size": 128000000000
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **"cname"** (*object[]*) – A list of objects containing SSD inventory information, one for each attached SSD. Note the label of this attribute will not be "cname" in the literal sense, but rather it will be the node's cname (eg, "c0-0c1s10n0")
- **"cname"[].bus** (*int*) – PCI bus to which the SSD is attached
- **"cname"[].device** (*int*) – SSD's PCI device number on the bus
- **"cname"[].func** (*int*) – SSD's PCI function identifier
- **"cname"[].model_number** (*string*) – SSD's model number
- **"cname"[].nid** (*int*) – NID number to which this SSD inventory information belongs
- **"cname"[].serial_number** (*string*) – SSD's serial number
- **"cname"[].size** (*int*) – Size of the SSD, not its remaining capacity
- **"cname"[].ssd_id** (*string*) – The SSD's PCI system device and vendor identifier
- **"cname"[].sub_id** (*string*) – The SSD's PCI sub-system device and vendor identifer

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

7.5 get_ssd_diags

The **get_ssd_diags** call provides third-party software with SSD diagnostic information. Diagnostic information is pulled from a database on the SMW. Information returned includes an object that has an attribute with a list of SSD

diagnostic objects as its value. Each SSD diagnostic object includes information such as a timestamp when the data was recorded in the database, life remaining, firmware version, serial_number, manufacturer's identifer, part identifier, PCI coordinates (bus, device, function), SSD size (not remaining capacity), and percentage used. Information may be returned for a targeted set of NIDs or all NIDs in the system. If there is no SSD diagnostic data in the databse for a targeted SSD, the returned information will have an empty list of SSD diagnostic objects. If an invalid NID is specified, an error response will be returned to the caller.

7.5.1 CLI Interface

capmc get_ssd_diags

-n, --nids <nidlist>
: Specify the NIDs for which to query SSD diagnositc information. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs. Cannot be used with -c.

-c, --cname <cnamelist>
: Specify a comma-separated list of node cnames for which to query SSD diagnositc information. If omitted, the default is all nodes. Cannot be used with -n.

```
$ capmc get_ssd_diags -c 'c0-0c0s4n2'
{
  "e": 0,
  "err_msg": "Success",
  "ssd_diags": [
    {
      "life_remaining": 100.0,
      "firmware": "8DV10170",
      "ts": "2016-06-23 09:55:25-06:00",
      "nid": 18,
      "serial_num": "CXA0000000000000-1",
      "cname": "c0-0c0s4n2",
      "manu_id": "8086",
      "percent_used": 0.0,
      "part_id": "8718",
      "comp_ord": "03:00:0",
      "size": 2000
    },
    ...
  ]
}
```

7.5.2 HTTP Interface

POST /capmc/get_ssd_diags
: **Example Request**:

```
POST /capmc/get_ssd_diags HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 27
Content-Type: application/json

{
  "nids": [
```

(continues on next page)

(continued from previous page)

```
    104
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes no arguments (ie, an empty object) or a single argument, which is either a list of NID numbers or a list of node cname strings.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs. This list must not contain invalid or duplicate NID numbers. If invalid NID numbers are specified then an error will be returned.
- **cname** (*string[]*) – User specified list, or empty array for all nodes. This list must not contain invalid cnames or cnames referencing objects that are not nodes. If invalid cnames are specified then an error will be returned.

Example Response:

JSON Parameters

- **e** (*int*) – Request status code, zero on success
- **err_msg** (*string*) – Human readable error message
- **ssd_diags** (*object[]*) – A list of objects containing SSD diagnostic information
- **ssd_diags[].cname** (*string*) – Cname of node to which this SSD diagnostic information belongs
- **ssd_diags[].comp_ord** (*string*) – The PCI bus, device, function coordinates
- **ssd_diags[].firmware** (*string*) – Firmware version flashed on SSD
- **ssd_diags[].life_remaining** (*float*) – SSD's remaining life
- **ssd_diags[].manu_id** (*string*) – Manufacturer's identifer
- **ssd_diags[].nid** (*int*) – NID number to which this SSD diagnostic information belongs
- **ssd_diags[].part_id** (*string*) – Manufacturer's part identifier
- **ssd_diags[].percent_used** (*float*) – Percentage of formatted size used
- **ssd_diags[].serial_number** (*string*) – SSD's serial number
- **ssd_diags[].size** (*int*) – SSD's size in GB, not its remaining capacity
- **ssd_diags[].ts** (*string*) – Timestamp when data was written to the database

Status Codes

- 200 OK – Network API call success
- 400 Bad Request – Bad request
- 500 Internal Server Error – Internal server error

CHAPTER

EIGHT

NODE ENERGY REPORTING

CAPMC API calls are provided such that external agents may query node energy consumption in various ways. The caller may query statistics by application id, job id, or simply use a list of NID numbers and a caller specified time window. Types of information returned may include aggregated energy usage on a set of nodes, energy usage per-node, or an energy accumulator point in time snapshot.

Warning: The **get_node_energy** and **get_node_energy_stats** API calls are resource intensive. Depending on system size and input parameters, those API calls may require several minutes to complete.

8.1 get_node_energy

Accumulated energy values for a set of nodes defined by a job (job_id), application (apid), list of NIDs (nids), or start and end time may be queried through the **get_node_energy** call. The input parameters are treated as conditions which are logically ANDed together. If an apid and start/end times are specified, then the values returned will be for the nodes involved in that apid during the interval specified by the start/end times.

Parameters returned include the following:

- Node count
- Duration of the interval, in seconds
- An array of (NID, energy) pairs

8.1.1 CLI Interface

capmc get_node_energy

-s, --start-time <YYYY-MM-DD HH:MM:SS>
Specify the starting time for the energy window calculation. This option must be used in conjunction with the **-e, –end-time** argument. If omitted, the starting time of the specified **apid** or **jobid** is used.

-e, --end-time <YYYY-MM-DD HH:MM:SS>
Specify the ending time for the energy window calculation. This option must be used in conjunction with the **-s, –start-time** argument. If omitted, the ending time of the specified **apid** or **jobid** is used.

-a, --apid <id>
Return statistics applicable to the NID list and application start & end times for the specified ALPS id.

-j, --job <id>
Return statistics applicable to the NID list and application start & end times for the specified batch scheduler job id.

-n, --nids <nidlist>
Specify the NIDs to use for the energy window calculation. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs matching other parameters.

8.1.2 HTTP Interface

POST /capmc/get_node_energy
Example Request:

```
POST /capmc/get_node_energy HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 122
Content-Type: application/json

{
  "start_time": "2015-06-03 14:07:32",
  "end_time": "2015-06-03 14:12:32",
  "nids": [
    23,
    24,
    25
  ]
}
```

Example Request: (By apid)

```
POST /capmc/get_node_energy HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 21
Content-Type: application/json

{
  "apid": 2977782
}
```

The request must **POST** a properly formatted JSON object to the API server. At a minimum, the request must contain a starting and ending time stamp with a NID list, an ALPS application ID, or a batch scheduler ID.

JSON Parameters

- **start_time** (*string*) – Optional, requested energy window sample start time
- **start_time** – Optional, requested energy window sample end time
- **nids** (*int[]*) – Optional, list of NIDs to use in energy counter query
- **apid** (*int*) – Optional, ALPS application ID
- **job_id** (*string*) – Optional, batch scheduler job id

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)


```
  "e": 0,
  "err_msg": "",
  "nid_count": 3,
  "time": 300.0,
  "nodes": [
    {
      "nid": 23,
      "energy": 62623
    },
    {
      "nid": 24,
      "energy": 45454
    },
    {
      "nid": 25,
      "energy": 42870
    }
  ]
}
```


JSON Parameters

- **e** (*int*) – Error status, non-zero indicates statistics are unavailable
- **err_msg** (*string*) – Human readable error string indicating failure reason
- **nid_count** (*int*) – Number of nodes used in statistics query
- **time** (*double*) – Window width of energy calculation, in seconds
- **nodes** (*object[]*) – Object array containing node level energy info, each element represents a single node
- **nodes[].nid** (*int*) – NID number owning the returned energy accumulation
- **nodes[].energy** (*int*) – Accumulated energy computed over the requested time interval, specified in Joules

Status Codes

- 200 OK – Network API call success
- 504 Gateway Timeout – Gateway Timeout
- 500 Internal Server Error – Internal command failure

8.2 get_node_energy_stats

Energy statistics for a set of nodes defined by a job (job_id), application (apid), list of NIDs (nodes), or start and end time may be queried through the **get_node_energy_stats** call. The input parameters are treated as conditions which are logically ANDed together. If an apid and start/end times are specified, then the statistics will be for the nodes involved in that apid during the interval specified by the start/end times. Both a temporal argument (apid, job_id, or start_time and end_time) and a component argument (apid, job_id, or NIDs) are required.

Parameters returned include the following:

- Total energy for the set
- Average energy for nodes in the set

- Standard deviation of energy for nodes in the set
- An ordered pair (NID, energy) for the minimum and maximum energy consuming nodes
- Duration of the interval, in seconds
- Node count

8.2.1 CLI Interface

capmc get_node_energy_stats

-s, --start-time <YYYY-MM-DD HH:MM:SS>
Specify the starting time for the energy window calculation. This option must be used in conjunction with the **-e, –end-time** argument. If omitted, the starting time of the specified **apid** or **jobid** is used.

-e, --end-time <YYYY-MM-DD HH:MM:SS>
Specify the ending time for the energy window calculation. This option must be used in conjunction with the **-s, –start-time** argument. If omitted, the ending time of the specified **apid** or **jobid** is used.

-a, --apid <id>
Return statistics applicable to the NID list and application start & end times for the specified APLS id.

-j, --job <id>
Return statistics applicable to the NID list and application start & end times for the specified batch scheduler job id.

-n, --nids <nidlist>
Specify the NIDs to use for the energy window calculation. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs matching other parameters.

8.2.2 HTTP Interface

POST /capmc/get_node_energy_stats
Example Request:

```
POST /capmc/get_node_energy_stats HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 122
Content-Type: application/json

{
  "start_time": "2015-06-03 14:07:32",
  "end_time": "2015-06-03 14:12:32",
  "nids": [
    23,
    24,
    25
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. At a minimum, the request must contain a starting and ending time stamp with a NID list, an ALPS application ID, or a batch scheduler ID.

JSON Parameters

- **start_time** (*string*) – Optional, requested energy window sample start time

- **start_time** – Optional, requested energy window sample end time
- **nids** (*int[]*) – Optional, list of NIDs to use in energy counter query
- **apid** (*int*) – Optional, ALPS application ID
- **job_id** (*string*) – Optional, batch scheduler job id

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "time": 300.0,
  "nid_count": 3,
  "energy_total": 150947,
  "energy_avg": 50315.666666666664,
  "energy_std": 8766.303072307936,
  "energy_max": [
    23,
    62623
  ],
  "energy_min": [
    25,
    42870
  ]
}
```

JSON Parameters

- **e** (*int*) – Error status, non-zero indicates statistics are unavailable
- **err_msg** (*string*) – Human readable error string indicating failure reason
- **time** (*double*) – Window width of energy statistics calculation, in seconds
- **energy_total** (*int*) – Sum of per node energy, in Joules
- **energy_avg** (*double*) – Per node average energy, in Joules
- **energy_std** (*double*) – Standard deviation, in Joules
- **energy_max** (*int[]*) – Ordered list identifying NID number followed by maximum observed node energy consumption
- **energy_min** (*int[]*) – Ordered list identifying NID number followed by minimum observed node energy consumption

Status Codes

- 200 OK – Network API call success
- 504 Gateway Timeout – Gateway Timeout
- 500 Internal Server Error – Internal command failure

8.3 get_node_energy_counter

Energy counters for a set of nodes defined by a job (job_id), application (apid), or list of NIDs (nids) may be queried through the **get_node_energy_counter** call. The parameters apid, jobid, and nids are treated as selectors for the set of nodes to query. If an apid or jobid are supplied, the running counters for each node in that aprun or job will be returned. If a list of NIDs is supplied, then the counters for the nodes corresponding to the supplied NID list will be returned. One and only one of apid, jobid, or NIDs list must be specified. If a time value is specified, then the query will retrieve the energy counters at or very near the specified time (if available, within one second). Otherwise, the most recent energy counter value will be returned.

Note: This API call returns a free running energy counter for each of the target NIDs. In order to be meaningful, such as when computing average power or energy consumed over a time interval, multiple calls must be made such that the caller can perform calculations based on the difference in returned energy counter values.

8.3.1 CLI Interface

capmc get_node_energy_counter

-t, --time <YYYY-MM-DD HH:MM:SS>
Specify the desired energy sample point in time. If omitted, the energy point in time will be taken as the most recent available sample in the last 30 seconds on a per node basis.

-a, --apid <id>
Return energy counters applicable to the NID list of the specified APLS id.

-j, --job <id>
Return energy counters applicable to the NID list of the specified batch scheduler job id.

-n, --nids <nidlist>
Specify the NIDs for which to retrieve enery counters. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs matching other parameters.

8.3.2 HTTP Interface

POST /capmc/get_node_energy_counter
Example Request:

```
POST /capmc/get_node_energy_counter HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 78
Content-Type: application/json

{
  "time": "2015-06-03 14:07:32",
  "nids": [
    23,
    24,
    25
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. At a minimum, the request must contain a time stamp with a NID list, an ALPS application ID, or a batch scheduler ID.

JSON Parameters

- **time** (*string*) – Optional, requested energy sample start time
- **nids** (*int[]*) – Optional, list of NIDs to use in energy counter query
- **apid** (*int*) – Optional, ALPS application ID
- **job_id** (*string*) – Optional, batch scheduler job id

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "nid_count": 3,
  "nodes": [
    {
      "nid": 24,
      "energy_ctr": 14802226,
      "time": "2015-06-03 14:07:32.886126-05"
    },
    {
      "nid": 23,
      "energy_ctr": 10196418,
      "time": "2015-06-03 14:07:32.022648-05"
    },
    {
      "nid": 25,
      "energy_ctr": 13649114,
      "time": "2015-06-03 14:07:32.886126-05"
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Error status, non-zero indicates statistics are unavailable
- **err_msg** (*string*) – Human readable error string indicating failure reason
- **nid_count** (*int*) – Number of nodes used in statistics query
- **nodes** (*object[]*) – Object array containing node level energy info, each element represents a single node
- **nodes[].nid** (*int*) – NID number owning the returned energy counter
- **nodes[].energy_cntr** (*int*) – Point in time energy accumulator value, specified in Joules
- **nodes[].time** (*string*) – Time stamp of returned energy value, includes fractional seconds and timezone offset

Status Codes

- 200 OK – Network API call success

- 504 Gateway Timeout – Gateway Timeout
- 500 Internal Server Error – Internal command failure

CHAPTER

NINE

SYSTEM LEVEL MONITORING

CAPMC API calls are provided such that external agents may monitor near real time system level power consumption and energy usage. If a time window is specified, then historical records may be retrieved. Data may be returned in aggregate or constituent form containing information relating to total system or per-cabinet components, respectively. Additionally, a mechanism is provided for a system administrator to convey intent or other operational parameters, such as a maximum system power limit, or unreported static power overhead, to third-parties.

9.1 get_system_parameters

Read-only parameters such as expected worst case system power consumption, static power overhead, or administratively defined values such as a system wide power limit, maximum power ramp rate, and target power band may be returned via the **get_system_parameters** call. Returned values are used to convey intent between the system administrator and external agents with respect to target power limits and other operational parameters. The returned parameters are strictly informational.

9.1.1 CLI Interface

capmc get_system_parameters

The command line interface has no arguments. Result text is returned via standard out.

```
$ capmc get_system_parameters
{
  "e": 0,
  "err_msg": "",
  "power_cap_target": 0,
..
```

9.1.2 HTTP Interface

POST /capmc/get_system_parameters

Example Request:

```
POST /capmc/get_system_parameters HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 2
Content-Type: application/json

{}
```

The request must **POST** an empty JSON object to the API server. This command takes no arguments.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "power_cap_target": 0,
  "power_threshold": 0,
  "static_power": 10700,
  "ramp_limited": false,
  "ramp_limit": 2000000,
  "power_band_min": 1000000,
  "power_band_max": 2000000
}
```

JSON Parameters

- **e** (*int*) – Error status, non-zero indicates failure
- **err_msg** (*string*) – Human readable error string indicating failure reason
- **power_cap_target** (*int*) – Administratively defined upper limit on system power
- **power_threshold** (*int*) – System power level, which if crossed, will result in Cray management software emitting over power budget warnings
- **static_power** (*int*) – Additional static system wide power overhead which is unreported, specified in watts
- **ramp_limited** (*bool*) – true if out-of-band HSS power ramp rate limiting features are enabled
- **ramp_limit** (*int*) – Administratively defined maximum rate of change (increasing or decreasing) in system wide power consumption, specified in watts per minute
- **power_band_min** (*int*) – Administratively defined minimum allowable system power consumption, specified in watts
- **power_band_max** (*int*) – Administratively defined maximum allowable system power consumption, specified in watts

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

Parameters including **ramp_limited**, **ramp_limit**, **power_band_min**, and **power_band_max** are new in SMW-8.0.

The **power_threshold** parameter is a mechanism in which a system administrator may convey intent to an energy aware scheduler. It defines a target, or a desired worst case system power usage. It is the responsibility of the scheduler to enforce the nessecary energy aware scheduling policy in order to comply with the administrators intent.

The **power_band_min** and **power_band_max** parameters allow an administrator to convey an external constraint to a workload manager. For example, a power utility company may state that a system should always consume a minimum amount of power, and not exceed a maximum amount of power. These values may be different than the system's minimum and maximum name plate rated power values.

9.2 get_system_power

The **get_system_power** call returns system level power information including the average, minimum, and maximum values observed over a user specified time interval. If no arguments are given, then information is returned for an interval consisting of the most recent 10 seconds.

9.2.1 CLI Interface

capmc get_system_power

-s, --start-time <YYYY-MM-DD HH:MM:SS>
Specify sampling window start time. If omitted, the default start time is 10 seconds into the past.

-w, --window-len <seconds>
Specify the sampling window length in seconds. The valid range is defined by the interval [2,3600]. If omitted, the default value of 10 seconds is used.

```
$ capmc get_system_power --start-time="2015-06-01 13:45:59" --window-len=30
{
  "start_time": "2015-06-01 13:45:59",
  "window_len": 30,
  "avg": 17488,
...
```

9.2.2 HTTP Interface

POST /capmc/get_system_power
Example Request:

```
POST /capmc/get_system_power HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 62
Content-Type: application/json

{
  "start_time": "2015-06-01 13:45:59",
  "window_len": 30
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes up to two optional arguments which identify a start time and window length.

JSON Parameters

- **start_time** (*string*) – Optional, sampling window start time
- **window_len** (*int*) – Optional, sampling window length

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
```

(continues on next page)

(continued from previous page)

```
    "err_msg": "",
    "window_len": 30,
    "start_time": "2015-06-01 13:45:59",
    "avg": 17488,
    "max": 17661,
    "min": 17340
}
```

JSON Parameters

- **e** (*int*) – Error status, non-zero indicates statistics are unavailable
- **err_msg** (*string*) – Human readable error string indicating failure reason
- **window_len** (*int*) – Window length in seconds in which the statistics have been computed, may be different from the requested value
- **start_time** (*string*) – Window sample start time in **YYYY-MM-DD HH:MM:SS** format or symbolic constant **CURRENT_TIMESTAMP**, may be different from the requested value
- **avg** (*int*) – Average system power computed over the time window
- **max** (*int*) – Peak system power observed over the time window
- **min** (*int*) – Min system power observed over the time window

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

9.3 get_system_power_details

The **get_system_power_details** call returns per cabinet power information including the average, minimum, and maximum values observed over a user specified time interval. If no arguments are given, then information is returned for an interval consisting of the most recent 10 seconds.

9.3.1 CLI Interface

capmc get_system_power_details

-s, --start-time <YYYY-MM-DD HH:MM:SS>
Specify sampling window start time. If omitted, the default start time is 10 seconds into the past.

-w, --window-len <seconds>
Specify the sampling window length in seconds. The valid range is defined by the interval [2,3600]. If omitted, the default value of 10 seconds is used.

```
$ capmc get_system_power_details --start-time="2015-06-01 13:45:59" --window-len=30
{
  "e": 0,
  "err_msg": "",
  "window_len": 30,
  "start_time": "2015-06-03 12:47:07",
```

(continues on next page)

(continued from previous page)

```
    "cabinets": [
      {
        "avg": 17432.033333333333,
        "max": 17730,
...
```

9.3.2 HTTP Interface

POST /capmc/get_system_power_details

Example Request:

```
POST /capmc/get_system_power_details HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 62
Content-Type: application/json

{
  "start_time": "2015-06-03 12:47:07",
  "window_len": 30
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes up to two optional arguments which identify a start time and window length.

JSON Parameters

- **start_time** (*string*) – Optional, sampling window start time
- **window_len** (*int*) – Optional, sampling window length

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "window_len": 30,
  "start_time": "2015-06-03 12:47:07",
    "cabinets": [
      {
        "avg": 17432.033333333333,
        "max": 17730,
        "min": 17069,
        "x": 0,
        "y": 0
      },
      {
        "avg": 17456.033333333333,
        "max": 17738,
        "min": 17060,
        "x": 1,
        "y": 0
      }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

JSON Parameters

- **e** (*int*) – Error status, non-zero indicates statistics are unavailable
- **err_msg** (*string*) – Human readable error string indicating failure reason
- **window_len** (*int*) – Window length in seconds in which the statistics have been computed, may be different from the requested value
- **start_time** (*string*) – Window sample start time in **YYYY-MM-DD HH:MM:SS** format or symbolic constant **CURRENT_TIMESTAMP**, may be different from the requested value
- **cabinets** (*object[]*) – Object array containing cabinet level power statistics, each element represents a single cabinet
- **cabinets[].avg** (*double*) – Average cabinet power computed over the time window
- **cabinets[].max** (*int*) – Peak cabinet power observed over the time window
- **cabinets[].min** (*int*) – Min cabinet power observed over the time window
- **cabinets[].x** (*int*) – Cabinet X coordinate, column address
- **cabinets[].y** (*int*) – Cabinet Y coordinate, row address

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

CHAPTER

TEN

UTILITY FUNCTIONS

CAPMC contains several utility functions which may be used to query information such as node to component name mapping, system partition membership, or a node's role assignment. The client utility may also function in an HTTP pass-through mode, effectively allowing a caller to specify custom input payloads not possible using command line arguments alone. This mode of operation offers maximum flexibility while allowing the caller to utilize capmc's built in authorization mechanism.

10.1 get_nid_map

Some commands, specifically commands relating to power capping controls, require a NID list which does not contain service nodes. Other use cases may require associating a geographic component name to NID number. In such cases, calling applications may query a node's role or cname using the **get_nid_map** call. The call returns a list objects where each element represents a single node. Each object in the list contains a numeric id identifying the NID number, geographic component name, and the operational role.

10.1.1 CLI Interface

capmc get_nid_map

-n, --nids <nidlist>
: Specify the NIDs to retreive NID number to component name mapping and assigned role. The syntax allows a comma-separated list of nids (eg, "1,4,5"), a range of nids (eg, "7-10"), or both (eg, "1,4,5,7-10"). If omitted, the default is all NIDs.

```
$ capmc get_nid_map --nids=1,140,141
{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "cname": "c0-0c2s3n1",
      "nid": 141,
      "role": "compute"
    },
    {
 ...
```

10.1.2 HTTP Interface

POST /capmc/get_nid_map

Example Request:

```
POST /capmc/get_nid_map HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 45
Content-Type: application/json

{
  "nids": [
    1,
    140,
    141
  ]
}
```

The request must **POST** a properly formatted JSON object to the API server. The command takes a single argument which identifies a target NID list.

JSON Parameters

- **nids** (*int[]*) – User specified list, or empty array for all NIDs.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "nids": [
    {
      "cname": "c0-0c2s3n1",
      "nid": 141,
      "role": "compute"
    },
    {
      "cname": "c0-0c2s3n0",
      "nid": 140,
      "role": "compute"
    },
    {
      "cname": "c0-0c0s0n1",
      "nid": 1,
      "role": "service"
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Error status, non-zero indicates failure
- **err_msg** (*string*) – Human readable error string

- **nids** (*object[]*) – Object array containing node specific mapping information, each element represents a single NID
- **nids[].nid** (*int*) – NID number owning the returned geographical component name and service role
- **nids[].cname** (*string*) – Geographical component name
- **nids[].role** (*string*) – Currently assigned service role, may be one of "compute" or "service"

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

10.2 get_partition_map

Some commands, specifically commands relating to frequency and sleep state controls, require a NID list which does not cross system partition boundaries. In such cases, calling applications may query partition membership using the **get_partition_map** call. The call returns a list objects where each element represents a single system partition. Each object in the list contains a numeric id identifying the partition number and the corresponding member NIDs.

10.2.1 CLI Interface

capmc get_partition_map

The command line interface has no arguments. Result text is returned via standard out.

```
$ capmc get_partition_map
{
  "e": 0,
  "err_msg": "",
  "partitions": [
    {
      "nids": [
        0,
        3,
..
```

10.2.2 HTTP Interface

POST /capmc/get_partition_map

Example Request:

```
POST /capmc/get_partition_map HTTP/1.1
Host: smw.example.com
Accept: application/json
Content-Length: 2
Content-Type: application/json

{}
```

The request must **POST** an empty JSON object to the API server. This command takes no arguments.

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "e": 0,
  "err_msg": "",
  "partitions": [
    {
      "partition": 0,
      "nids": [
        0,
        3,
        166,
        167
      ]
    }
  ]
}
```

JSON Parameters

- **e** (*int*) – Error status, non-zero indicates failure
- **err_msg** (*string*) – Human readable error string
- **partitions** (*object[]*) – Object array containing partition membership information, each element represents a single system partition
- **partitions[].partition** (*int*) – Partition number owning the returned NID membership list
- **partitions[].nids** (*int[]*) – Partition member NID list

Status Codes

- 200 OK – Network API call success
- 500 Internal Server Error – Internal command failure

10.3 json

The client script, capmc, provides a function which allows a caller to construct and send a JSON formatted object to a user specified API handler. This command allows a caller to utilize the capmc authorization mechanism while constructing their own input parameter objects directly. This command is implemented purely within the client side script.

10.3.1 CLI Interface

capmc json

-r, --resource </capmc/api/path>
Post a JSON text data structure acquired from standard input to the specified resource on the server. JSON text input is limited to 10MB.

```
$ capmc json --resource=/capmc/node_status < /path/to/input/data.json
```

APPENDIX

A

GLOSSARY

ALPS Application Level Placement Scheduler

API Application Programming Interface

ASIC Application Specific Integrated Circuit

CAPMC Cray Advanced Platform Monitoring & Control

CA Certificate Authority

CLI Command Line Interface

C-State Processor idle state

HTTP HyperText Transfer Protocol

HSN High Speed Network

HSS Hardware Supervisory Subsystem

JSON JavaScript Object Notation

NID Node identifier number

POST HTTP transfer method

P-State Processor performance state

SMW System Management Workstation

X.509 Cryptographic industry standard for public key infrastructure

HTTP ROUTING TABLE

/capmc

POST /capmc/clr_mcdram_cfg, 54
POST /capmc/clr_numa_cfg, 64
POST /capmc/clr_power_bias, 24
POST /capmc/clr_ssd_enable, 71
POST /capmc/cnctl, 44
POST /capmc/compute_power_bias, 26
POST /capmc/get_mcdram_capabilities, 48
POST /capmc/get_mcdram_cfg, 50
POST /capmc/get_nid_map, 92
POST /capmc/get_node_energy, 78
POST /capmc/get_node_energy_counter, 82
POST /capmc/get_node_energy_stats, 80
POST /capmc/get_node_rules, 3
POST /capmc/get_node_status, 6
POST /capmc/get_numa_capabilities, 58
POST /capmc/get_numa_cfg, 60
POST /capmc/get_partition_map, 93
POST /capmc/get_power_bias, 22
POST /capmc/get_power_cap, 16
POST /capmc/get_power_cap_capabilities, 14
POST /capmc/get_ssd_diags, 75
POST /capmc/get_ssd_enable, 68
POST /capmc/get_ssds, 73
POST /capmc/get_system_parameters, 85
POST /capmc/get_system_power, 87
POST /capmc/get_system_power_details, 89
POST /capmc/node_off, 9
POST /capmc/node_on, 8
POST /capmc/node_reinit, 10
POST /capmc/set_mcdram_cfg, 52
POST /capmc/set_numa_cfg, 62
POST /capmc/set_power_bias, 23
POST /capmc/set_power_bias_data, 25
POST /capmc/set_power_cap, 19
POST /capmc/set_ssd_enable, 70

INDEX

Symbols

-A, –accel <watts>
  set_power_cap command line option, 19
-M, –max <max frequency>
  set_freq_limits command line option, 35
-N, –node <watts>
  set_power_cap command line option, 19
-a, –apid <id>
  get_node_energy command line option, 77
  get_node_energy_counter command line option, 82
  get_node_energy_stats command line option, 80
-c, –cname <cnamelist>
  get_ssd_diags command line option, 75
  get_ssds command line option, 72
-e, –end-time <YYYY-MM-DD HH:MM:SS>
  get_node_energy command line option, 77
  get_node_energy_stats command line option, 80
-f, –filter <opts>
  node_status command line option, 5
-j, –job <id>
  get_node_energy command line option, 77
  get_node_energy_counter command line option, 82
  get_node_energy_stats command line option, 80
-l, –limit <sleep state limit>
  set_sleep_state_limit command line option, 43
-m, –min <min frequency>
  set_freq_limits command line option, 35
-m, –mode <a2a|snc2|snc4|hemi|quad|reset>
  set_numa_cfg command line option, 62
-m, –mode <enable|disable|reset>
  set_ssd_enable command line option, 69
-m, –mode <flat|0|split|25|equal|50|cache|100|reset>
  set_mcdram_cfg command line option, 52
-n, –nids <nidlist>
  get_freq_capabilities command line option, 29
  get_freq_limits command line option, 32
  get_mcdram_capabilities command line option, 47
  get_mcdram_cfg command line option, 50
  get_nid_map command line option, 91
  get_node_energy command line option, 77
  get_node_energy_counter command line option, 82
  get_node_energy_stats command line option, 80
  get_numa_capabilities command line option, 57
  get_numa_cfg command line option, 60
  get_power_cap command line option, 16
  get_power_cap_capabilities command line option, 13
  get_sleep_state_limit command line option, 41
  get_sleep_state_limit_capabilities command line option, 38
  get_ssd_diags command line option, 75
  get_ssd_enable command line option, 67
  get_ssds command line option, 72
  node_off command line option, 9
  node_on command line option, 7
  node_reinit command line option, 10
  node_status command line option, 5
  set_freq_limits command line option, 35
  set_mcdram_cfg command line option, 52
  set_numa_cfg command line option, 62
  set_power_cap command line option, 19
  set_sleep_state_limit command line option, 43
  set_ssd_enable command line option, 69
-r, –reason <log message>
  node_off command line option, 9
  node_on command line option, 7
  node_reinit command line option, 10
-r, –resource </capmc/api/path>
  json command line option, 94
-s, –start-time <YYYY-MM-DD HH:MM:SS>
  get_node_energy command line option, 77
  get_node_energy_stats command line option, 80
  get_system_power command line option, 87
  get_system_power_details command line option, 88
-t, –time <YYYY-MM-DD HH:MM:SS>
  get_node_energy_counter command line option, 82
-w, –window-len <seconds>
  get_system_power command line option, 87
  get_system_power_details command line option, 88

A

ALPS, **97**
API, 1, **97**
ASIC, **97**

C

C-State, **97**
CA, **97**
CAPMC, **97**
capmc, 1
CLI, **97**

E

environment variable
  OS_CACERT, 2
  OS_CERT, 2
  OS_KEY, 2
  OS_SERVICE_URL, 2

G

get_freq_capabilities command line option
  -n, –nids <nidlist>, 29
get_freq_limits command line option
  -n, –nids <nidlist>, 32
get_mcdram_capabilities command line option
  -n, –nids <nidlist>, 47
get_mcdram_cfg command line option
  -n, –nids <nidlist>, 50
get_nid_map command line option
  -n, –nids <nidlist>, 91
get_node_energy command line option
  -a, –apid <id>, 77
  -e, –end-time <YYYY-MM-DD HH:MM:SS>, 77
  -j, –job <id>, 77
  -n, –nids <nidlist>, 77
  -s, –start-time <YYYY-MM-DD HH:MM:SS>, 77
get_node_energy_counter command line option
  -a, –apid <id>, 82
  -j, –job <id>, 82
  -n, –nids <nidlist>, 82
  -t, –time <YYYY-MM-DD HH:MM:SS>, 82
get_node_energy_stats command line option
  -a, –apid <id>, 80
  -e, –end-time <YYYY-MM-DD HH:MM:SS>, 80
  -j, –job <id>, 80
  -n, –nids <nidlist>, 80
  -s, –start-time <YYYY-MM-DD HH:MM:SS>, 80
get_numa_capabilities command line option
  -n, –nids <nidlist>, 57
get_numa_cfg command line option
  -n, –nids <nidlist>, 60
get_power_cap command line option
  -n, –nids <nidlist>, 16
get_power_cap_capabilities command line option
  -n, –nids <nidlist>, 13
get_sleep_state_limit command line option
  -n, –nids <nidlist>, 41
get_sleep_state_limit_capabilities command line option
  -n, –nids <nidlist>, 38
get_ssd_diags command line option
  -c, –cname <cnamelist>, 75
  -n, –nids <nidlist>, 75
get_ssd_enable command line option
  -n, –nids <nidlist>, 67
get_ssds command line option
  -c, –cname <cnamelist>, 72
  -n, –nids <nidlist>, 72
get_system_power command line option
  -s, –start-time <YYYY-MM-DD HH:MM:SS>, 87
  -w, –window-len <seconds>, 87
get_system_power_details command line option
  -s, –start-time <YYYY-MM-DD HH:MM:SS>, 88
  -w, –window-len <seconds>, 88

H

HSN, **97**
HSS, **97**
HTTP, **97**

J

JSON, **97**
json command line option
  -r, –resource </capmc/api/path>, 94

N

NID, **97**
node_off command line option
  -n, –nids <nidlist>, 9
  -r, –reason <log message>, 9
node_on command line option
  -n, –nids <nidlist>, 7
  -r, –reason <log message>, 7
node_reinit command line option
  -n, –nids <nidlist>, 10
  -r, –reason <log message>, 10
node_status command line option
  -f, –filter <opts>, 5
  -n, –nids <nidlist>, 5

P

P-State, **97**
POST, **97**

R

RPM, 1

S

set_freq_limits command line option
  -M, –max <max frequency>, 35
  -m, –min <min frequency>, 35
  -n, –nids <nidlist>, 35
set_mcdram_cfg command line option

-m, –mode <flat|0|split|25|equal|50|cache|100|reset>, 52
-n, –nids <nidlist>, 52
set_numa_cfg command line option
-m, –mode <a2a|snc2|snc4|hemi|quad|reset>, 62
-n, –nids <nidlist>, 62
set_power_cap command line option
-A, –accel <watts>, 19
-N, –node <watts>, 19
-n, –nids <nidlist>, 19
set_sleep_state_limit command line option
-l, –limit <sleep state limit>, 43
-n, –nids <nidlist>, 43
set_ssd_enable command line option
-m, –mode <enable|disable|reset>, 69
-n, –nids <nidlist>, 69
SMW, 1, **97**

X

X.509, 2, **97**