Cloud Server Multi Node System Specifications

Software: Multi Node System Architecture

Date: July 6, 2015
Version: V0.7.5

Revision History

|  |  |
| --- | --- |
| Revision/Date | Notes |
| V0.7.1 | * Join Blade, Node and architecture specification
 |
| V0.7.2 | * Reduce command set
* Incorporate comment feedback.
 |
| V0.7.3 | * Add common OEM commands
 |
| V0.7.4 | * Update Node Standard API
 |
| V0.7.5 | * Remove IPMI references and apply IPMI 2.0 cross references
* Change formatting and alignment throughout
 |

Contents

[1. Design Requirements 8](#_Toc424143667)

[1.1 Event Log 8](#_Toc424143668)

[1.2 Power Control (On/Off/Reset/Cycle) 8](#_Toc424143669)

[1.3 Sensors (get/set) 9](#_Toc424143670)

[1.4 Firmware Update 9](#_Toc424143671)

[1.5 Serial Console Redirection 9](#_Toc424143672)

[1.6 Management LAN setup 10](#_Toc424143673)

[1.7 FRU Dump 10](#_Toc424143674)

[1.8 Management Controller Information 10](#_Toc424143675)

[1.9 Power Reading 10](#_Toc424143676)

[1.10 Boot Status/Control 10](#_Toc424143677)

[1.11 Fan Speed Monitoring/Control 11](#_Toc424143678)

[1.12 Identify 11](#_Toc424143679)

[1.13 Management Interface Access 11](#_Toc424143680)

[1.14 Node/Spine Enumeration 12](#_Toc424143681)

[2. Multi Node Spine and Node Physical Architecture 13](#_Toc424143682)

[3. Multi Node Management System 14](#_Toc424143683)

[3.1 Multi Node Chassis/Rack Manager Architecture 14](#_Toc424143684)

[3.2 Multi Node Blade Architecture 15](#_Toc424143685)

[4. Spine API Overview 16](#_Toc424143686)

[4.1 Spine Manageability Command Set 17](#_Toc424143687)

[5. Spine Management Controller Requirements 18](#_Toc424143688)

[5.1 Interfaces 18](#_Toc424143689)

[5.2 Bus Support 18](#_Toc424143690)

[5.3 BMC Debug Port 18](#_Toc424143691)

[5.4 Serial Port Timeout 18](#_Toc424143692)

[5.5 Sensor Data Record 19](#_Toc424143693)

[5.6 System Event Log 19](#_Toc424143694)

[5.7 Firmware Decompression 19](#_Toc424143695)

[5.8 Node Power Control 19](#_Toc424143696)

[5.9 Node Present 19](#_Toc424143697)

[6. Spine Management 20](#_Toc424143698)

[6.1 Spine Serial Console Support 21](#_Toc424143699)

[6.2 Spine Serial Console Activation 22](#_Toc424143700)

[6.3 Command-Completion Codes 25](#_Toc424143701)

[7. Spine Command Formats 26](#_Toc424143702)

[7.1 Get Device ID 26](#_Toc424143703)

[7.2 Get System GUID 26](#_Toc424143704)

[7.3 Get Channel Authentication Capabilities 26](#_Toc424143705)

[7.4 Send Message 26](#_Toc424143706)

[7.5 Master Write-Read 26](#_Toc424143707)

[7.6 Chassis Identify 26](#_Toc424143708)

[7.7 Get Sensor Reading Factors 27](#_Toc424143709)

[7.8 Get Sensor Reading 27](#_Toc424143710)

[7.9 Get Sensor Type 27](#_Toc424143711)

[7.10 Read FRU Data 27](#_Toc424143712)

[7.11 For command implementation review Intel IPMI Specification section 34.2Write FRU Data 27](#_Toc424143713)

[7.12 For command implementation review Intel IPMI Specification section 34.3Get SDR 28](#_Toc424143714)

[7.13 Reserve SEL 28](#_Toc424143715)

[7.14 Get SEL Entry 29](#_Toc424143716)

[7.15 Add SEL Entry 29](#_Toc424143717)

[7.16 Clear SEL 30](#_Toc424143718)

[7.17 Get Power Reading 30](#_Toc424143719)

[7.18 Get Power Limit 30](#_Toc424143720)

[7.19 Set Power Limit 30](#_Toc424143721)

[7.20 Activate Power Limit 30](#_Toc424143722)

[7.21 StartSerialSession 30](#_Toc424143723)

[7.22 ReceiveSerialData 31](#_Toc424143724)

[7.23 SendSerialData 32](#_Toc424143725)

[7.24 StopSerialSession 33](#_Toc424143726)

[7.25 GET GPIO 33](#_Toc424143727)

[7.26 SET GPIO 33](#_Toc424143728)

[7.27 Get CPLD Info 34](#_Toc424143729)

[7.28 Write Firmware Update 34](#_Toc424143730)

[7.29 Read Firmware Update 35](#_Toc424143731)

[7.30 Firmware Update 36](#_Toc424143732)

[7.31 Get Nodes Present 36](#_Toc424143733)

[7.32 Sensor Data Repository 37](#_Toc424143734)

[8. System Event Logs 38](#_Toc424143735)

[9. Overview of the Node API 40](#_Toc424143736)

[9.1 Manageability Command Set 40](#_Toc424143737)

[10. Node Implementation Requirements 42](#_Toc424143738)

[10.1 Interfaces 42](#_Toc424143739)

[10.2 IPMI Bus Support 42](#_Toc424143740)

[10.3 System Event Log 42](#_Toc424143741)

[10.4 SMBus Slave Address 42](#_Toc424143742)

[10.5 GPIO Mapping 42](#_Toc424143743)

[10.6 Keyboard Controller Style (KCS) Interface 43](#_Toc424143744)

[10.7 Firmware Decompression 43](#_Toc424143745)

[10.8 Inlet Sensor 43](#_Toc424143746)

[10.9 Serial Console Support 43](#_Toc424143747)

[10.10 Node Serial Console Support 43](#_Toc424143748)

[10.11 Node Identification Commands 43](#_Toc424143749)

[11. Node Command Formats 44](#_Toc424143750)

[11.1 Get Device ID 44](#_Toc424143751)

[11.2 Get System GUID 44](#_Toc424143752)

[11.3 Send Message 44](#_Toc424143753)

[11.4 Master Write-Read 44](#_Toc424143754)

[11.5 Get Chassis Status 44](#_Toc424143755)

[11.6 Chassis Control 44](#_Toc424143756)

[11.7 Chassis Identify 45](#_Toc424143757)

[11.8 Set Power Restore Policy 45](#_Toc424143758)

[11.9 Set System Boot Options 45](#_Toc424143759)

[11.10 Get System Boot Options 45](#_Toc424143760)

[11.11 Get Sensor Reading 45](#_Toc424143761)

[11.12 Get Sensor Type 45](#_Toc424143762)

[11.13 Get FRU Inventory Area Info 45](#_Toc424143763)

[11.14 Read FRU Data 46](#_Toc424143764)

[11.15 Write FRU Data 46](#_Toc424143765)

[11.16 Reserve SDR Repository 46](#_Toc424143766)

[11.17 Get SDR 47](#_Toc424143767)

[11.18 Reserve SEL 47](#_Toc424143768)

[11.19 Get SEL Info 48](#_Toc424143769)

[11.20 Get SEL Entry 48](#_Toc424143770)

[11.21 Add SEL Entry 50](#_Toc424143771)

[11.22 Clear SEL 50](#_Toc424143772)

[11.23 Get Power Reading 50](#_Toc424143773)

[11.24 Get Power Limit 50](#_Toc424143774)

[11.25 Set Power Limit 50](#_Toc424143775)

[11.26 Activate Power Limit 50](#_Toc424143776)

[11.27 Send Request Message to BMC 51](#_Toc424143777)

[11.28 Get All GPIO Status 51](#_Toc424143778)

[11.29 Set All GPIO Status 52](#_Toc424143779)

[11.30 Get GPIO Configuration 52](#_Toc424143780)

[11.31 Set GPIO Configuration 54](#_Toc424143781)

[11.32 Send Interrupt to BMC 54](#_Toc424143782)

[11.33 Get BIOS Code 55](#_Toc424143783)

[11.34 Firmware Update 56](#_Toc424143784)

[11.35 Firmware Verify 56](#_Toc424143785)

[11.36 Get Firmware Version 57](#_Toc424143786)

[12. Sensor Data Repository 58](#_Toc424143787)

[13. Appendix 1: Reference Material 59](#_Toc424143788)

[13.1 Sensor Types 59](#_Toc424143789)

[13.2 Event Reading Codes 59](#_Toc424143790)

[13.3 Sensor Unit Type Codes 59](#_Toc424143791)

[13.4 Entity IDs 59](#_Toc424143792)

[14. System Event Logs 60](#_Toc424143793)

[14.1 System Event Logs Generated by BMC 60](#_Toc424143794)

[14.2 System Event Logs Generated by BIOS 60](#_Toc424143795)

[14.2.1 Memory ECC Error Logging 60](#_Toc424143796)

[14.2.2 PCI Express Error Logging 62](#_Toc424143797)

[15. Appendix: Commonly Used of Acronyms 67](#_Toc424143798)

# Design Requirements

## Event Log

**[EVENT.LOG1.0]** The multi-node hardware must be capable of storing event logs for each node in the multi-node system.

**[EVENT.LOG2.0]** The multi-node system should associate each log entry to the corresponding node to which the entry belongs.

**[EVENT.LOG3.0]** The log storage should be an aggregation of all nodes or on a per node basis.

**[EVENT.LOG4.0]** it should be possible to clear the log at node level granularity.

**[EVENT.LOG5.0]** logically the event log provide a granularity at node level and physically the event log can be stored at a minimum granularity of FRU.

*For example the vendor can choose to physically store the logs at each node level or at the FRU to which the node belongs which could be sled (where one or more nodes are connected).*

**[EVENT.LOG6.0]** It should be possible to associate each SEL entry to a particular node.

*In the event the nodes are moved within the chassis it would still be possible to identify which nodes those log entries were originated. For example the UUID of the node to which the log entry belongs to could be stored with the log.*

**[EVENT.LOG7.0]** UTC timestamp should be stored with each entry to identify the time of the entry.

## Power Control (On/Off/Reset/Cycle)

**[POWER.CTRL1.0]** It should be possible to power control or reset each node.(add a separate section for BMC)

## Sensors (get/set)

**[SENSOR1.0]** It should be possible to get the sensors setting for all required sensors (described later in this document) that are in system.

**[SENSOR2.0]** It should be possible to get thresholds, location and unit for each sensor.

**[SENSOR3.0]** It is desirable to be able to set the threshold values for user configurable settings for the sensors.

**[SENSOR4.0]** It is desirable to be able to reset the changed threshold values to factory default values.

## Firmware Update

**[FW.UPDATE1.0]** It should be possible to remotely update the firmware for the management controller and Node firmware (Bios or pre-boot environment firmware).

**[FW.UPDATE2.0]** It is desirable to have rest of the firmware upgrades also applied remotely, however if the feature not available, these should be possible to apply from within the Operating System.

## Serial Console Redirection

**[CONSOLE.REDIRECT1.0]** It should be possible to redirect serial console of each managed node over the network.

**[CONSOLE.REDIRECT2.0]** The redirection should be possible over a dedicated network link or a shared network connection with the operating system.

**[CONSOLE.REDIRECT3.0]** It is desirable to be able to redirect serial connections for all nodes, however, it should be possible to redirect more than one concurrent serial connections and should be possible to query that number and the number of active sessions.

## Management LAN setup

**[MGMT.LAN1.0]** It should be possible to both get and set the management LAN parameters like ipv4, ipv6, hostname, netmask, dns servers etc.

**[MGMT.LAN2.0]** It must be possible to setup dhcpv4 and dhcpv6 on the management LAN.

## FRU Dump

**[FRU.DATA1.0]** It should be possible to dump the FRU details including asset, serial, revision etc.

*For the complete list of parameters to dump please refer to microserver specification.*

## Management Controller Information

**[MGMT\_CTRL1.0]** It should be possible to query management controller information like status, version etc.

## Power Reading

**[POWER1.0]** It should be possible to query power at chassis, sled and node level.

*[Need more discussion on this as power reading at node level may require inrush controllers that would add to the cost]*

**[POWER 2.0]** Power capping should be implemented at node and sled level. Higher level (at chassis level) may be desirable but would be implementation dependent and is out of scope for this discussion. [ Need to revisit and confirm if this feature is required]

## Boot Status/Control

**[BOOT.STATUS1.0]** should be possible to query current boot order and be able to control/change it. This should be implemented at node level.

## Fan Speed Monitoring/Control

**[FAN1.0]** It should be possible to query fan speed.

**[FAN2.0]** It should be possible to set fan speed and algorithm (like pwm etc).

**[FAN3.0]** Fan speed status and control should be implemented at same level where fans are connected.

**[FAN4.0]** If power capping is available, then capability to ensure fan speed range requirement over a range of temperature is included in the power capping or specify inlet temperature over which the power capping is guaranteed.

## Identify

**[UID1.0]** It should be possible to turn the identification light and off.

**[UID2.0]** The Identity LED status and control should be implemented at chassis and FRU level.

*e.g. if sled is field replaceable, then there should be identification light at sled.*

## Management Interface Access

**[MANAGEMENT.INTERFACE1.0]** Access to management interface could be dedicated network interface or NC-SI.

**[MANAGEMENT.INTERFACE2.0]** From local machine at least one common interface from these (define 3 interfaces here. KCS etc) should be available.

**[MANAGEMENT.INTERFACE3.0]** Management interface access should be consistent across multiple vendors and multiple platforms (single node, multi-node and microservers).

**[MANAGEMENT.INTERFACE4.0]** It is desirable that the access be ReSTful.

## Node/Spine Enumeration

**[ENUMERATION1.0]** It should be possible to enumerate the nodes/sleds in the system both remotely and from the local node (local node should be able to query where it is located on the chassis).

*[This need to be done consistently across vendors, need to define a common enumeration spec]*

# Multi Node Spine and Node Physical Architecture

The spine contains multiple child nodes, the Spine Baseboard Management Controller (spine-BMC) communicates with the Node Baseboard Management Control (node-BMC) using I2C. The spine has GPIO defined to determine node presence, node power control and ALERT signal.



# Multi Node Management System

The multi node management systems discussed in this specification consist of the following architectures:

1. Rack Manager, Spine, Nodes
2. Chassis Manager Spine, Nodes
3. Network attached Spine, Nodes

## **Multi Node Chassis/Rack** Manager Architecture

The Chassis/Rack Manager Spine and Node interfaces is exposed through REST. The REST interface supports the following security mechanisms: TLS/SSL, IPSEC and Kerberos Authentication.



## Multi Node Blade Architecture

The network connected blade spine and node interfaces can be exposed through REST/IPMI. The REST interface supports SSL encryption. An IPMI interface would support RMCP+.



A single physical network controller on the blade spine management controller should support multiple channels with unique MAC and IP assignments to individual nodes.



# Spine API Overview

The multi node blade spine protocol is a small subset of the intelligent platform management interface (IPMI 2.0) protocol.

If the spine baseboard management controller communicates over Ethernet directly, then the spine protocol should conform to IPMI 2.0 with SSL and REST interface support.

If a Chassis Manager is present, that communication with the blade spine management controller firmware over serial, the communication will IPMI basic mode over the IPMI Serial/Modem Interface. Reference: IPMI 2.0 - 14.4 Basic Mode. Full IPMI command and interface conformance is not required.

Note: For the purpose of completeness this document contains references and abstracts from the IPMI specification: <http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html>

If the blade spine is network accessible, then the communication with be IPMI over LAN using RMCP+ or REST. In this instance, IPMI network session configuration and support for, user authentication, encryption and confidentiality should be supported.

Table 1 lists the IPMI commands that are required or optional for compute blades.

## Spine Manageability Command Set

Table . Required IPMI Commands for Blades

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Command name** | **Reference** | **Type** | **Fn** | **Cmd** |
| Get Device ID  | 20.1 | App | 06h | 01h |
| Get System GUID  | 22.14 | App | 06h | 37h |
| Get Channel Authentication Capabilities | 22.13 | App | 06h | 38h |
| Send Message | 22.7 | App | 06h | 34h |
| Master Write-Read | 22.11 | App | 06h | 52h |
| Get Chassis Status  | 28.2 | Chassis | 00h | 01h |
| Chassis Control  | 28.3 | Chassis | 00h | 02h |
| Chassis Identify  | 28.5 | Chassis | 00h | 04h |
| Get Sensor Reading  | 35.14 | Sensor | 04h | 2Dh |
| Get Sensor Type | 35.16 | Sensor | 04h | 2FH |
| Read FRU Data  | 34.2 | Storage | 0Ah | 11h |
| Write FRU Data  | 34.3 | Storage | 0Ah | 12h |
| Get SDR  | 33.12 | Storage | 0Ah | 23h |
| Reserve SEL  | 31.4 | Storage | 0Ah | 42h |
| Get SEL Entry  | 31.5 | Storage | 0Ah | 43h |
| Add SEL Entry | 31.6 | Storage | 0Ah | 44h |
| Clear SEL  | 31.9 | Storage | 0Ah | 47h |
| Get Power Reading | N/A | DCMI | 2Ch | 02h |
| Get Power Limit | N/A | DCMI | 2Ch | 03h |
| Set Power Limit | N/A | DCMI | 2Ch | 04h |
| Activate Power Limit | N/A | DCMI | 2Ch | 05h |
| StartSerialSession  | N/A | OEM | 30h | 2Bh |
| ReceiveSerialData | N/A | OEM | 30h | 2Dh |
| SendSerialData | N/A | OEM | 30h | 2Eh |
| StopSerialSession | N/A | OEM | 30h | 2Fh |
| Get GPIO | N/A | OEM | 30h | E0h |
| Set GPIO | N/A | OEM | 30h | E1h |
| Get CPLD Version | N/A | OEM | 30h | 17h |
| Write Firmware Update | N/A | OEM | 30h | 81h |
| Read Firmware Update | N/A | OEM | 30h | 82h |
| Firmware Update | N/A | OEM | 30h | 83h |
| Get Nodes Present | N/A | OEM | 30h | 80h |

*M**= Mandatory, N/A**= Not applicable to blade type*

# Spine Management Controller Requirements

The following sections describe the Spine implementation requirements.

Only a small subset of commands of IPMI2.0 is required to function as a blade spine for multi nodes.

## Interfaces

* REST SSL
* IPMI over LAN (RMCP+)
* Serial Interface
* I2C

If the spine management controller is network accessed it should support REST and optionally IPMI over LAN. If the management controller is accessed by serial it should support the Serial Interface.

## Bus Support

* SMBus (I2C)
* PHY/NCSI – LAN (I2C)
* Serial - IPMI Basic Mode
* Serial Peripheral Interface (SPI)

## BMC Debug Port

## Serial Port Timeout

If the blade spine communicates with a Chassis Manager over serial line, then the receiving transmission timeout of 100 milliseconds (ms) should be strictly implemented. The spine BMC must therefore respond to all serial IPMI requests within 100ms. When it receives a response from the baseboard management controller (BMC), the Chassis Manager might immediately send another request. It is therefore possible that the BMC will receive multiple requests within a 100ms timeframe, depending on its response time.

## Sensor Data Record

When enumerating the SDR, the first record accessed (0000) must always be the PWM. It must also have the sensor number 01.

The multi node blade BMC is responsible for polling all nodes and reporting a single PWM for all nodes. The PWM reported should be the highest value of the aggregate node.

## System Event Log

The System Event Log (SEL) should implement circular logging. When the SEL is full, the log should overwrite the oldest record with the newest record.

The spine SEL should be capable for storing spine events and propagating them to the impacted nodes with the Add SEL entry command.

## Firmware Decompression

The blade spine BMC firmware must deterministically be fully decompressed (loaded) and ready to service IPMI request messages within xx seconds.

## Node Power Control

The spine BMC SET/GET GPIO command will support the node Power Enable, which will control power to each node.

## Node Present

The spine BMC will have dedicated GPIO to determine whether nodes are present. The GET GPIO will be sent directly to the blade to identify if the node present GPIO is active.

# Spine Management

Multi-Node Blade Spine Management Controller refers to the “Management Controller” listed in the diagram:

In typical operation a significant portion of the commands received by the blade spine management controller will be Send Message commands carrying payload destined for a the node management controller. The spine management controller will act as a proxy I2C controller forwarding messages between remote hosts and nodes. The spine management controller will perform all the I2C signaling, and IPMI request/response message encapsulation.

## **Spine Serial Console Support**

The blade spine management controller should support control logic that enables it to monitor all node management controller serial console output.

The spine management controller is required to support reading, buffering and writing to all node console ports individually. The spine management controller acts as a caching buffer between the remote host or rack/chassis manager incoming IPMI requests message and the node console output response message, encapsulating console IO into IPMI messages.

To support serial console redirection, the management controller makes use of four OEM IPMI commands:

StartSerialSession

*Instructs the management controller to start polling the node system console serial port and buffering the data.*

**ReceiveSerialData**

*Remote host or Chassis Manager will send this in a loop, this command reads the serial console data from the management controller buffer. The management controller will encapsulate the node serial console buffer into an IPMI response command, flush the buffer and send the response to the Remote host or Chassis Manager.*

**SendSerialData**

*Spine management controller extracts the encapsulated serial console payload from the IPMI message and writes data to the node serial port.*

**StopSerialSession**

*Spine management controller stops the polling thread activated by the StartSerialSession command against the target node.*

The table below lists the IPMI commands that are required for compute blades (Note that “M” is mandatory).

The commands are described in detail in the sections that follow.

Required IPMI Commands for Blades

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command name** | **Reference** | **Type** | **Fn** | **Cmd** | **Compute blade** |
| StartSerialSession  | N/A | OEM | 30h | 2Bh | M |
| ReceiveSerialData | N/A | OEM | 30h | 2Dh | M |
| SendSerialData | N/A | OEM | 30h | 2Eh | M |
| StopSerialSession | N/A | OEM | 30h | 2Fh | M |

## **Spine Serial Console Activation**

To obtain a serial console session the remote host or Remote Host or Chassis Manager will issue the StartSerialSession IPMI command encapsulated in the Send Message command. The spine management controller will start a thread to read serial console data from the node serial port (node is determined by the channel in the Send Message command) and copy the data into an internal buffer. The remote host or Chassis Manager may continually send the ReceiveSerialData command to the spine management controller. The management controller on the spine will encapsulate the node serial console buffer into an IPMI response message and send the response to the remote host or Chassis Manager.

When the remote host or Chassis Manager needs to interact with the node serial console, it will execute the SendSerialData command to the spine management controller. The management controller will strip the IPMI message framing and Send Message framing from the payload and forward the raw payload to the node serial port.

It is expected that the management controller forwarding of SendSerialData does not block the ReceiveSerialData thread. It is therefore anticipated that separate threads will be used.

When the remote host or Chassis Manager wishes to stop collecting serial console data for a given node, it will send the StopSerialSession to the spine management controller. The management controller will stop the node system console polling and buffering for the target node.

The StartSerialSession command includes a timeout parameter to inform the spine management controller to automatically stop the node serial port polling thread if no ReceiveSerialData or SendSerialData commands have been sent by the Chassis Manager for a given timeout period.

The StartSerialSession command will also include a flush buffer parameter, which informs the spine management controller to flush the node serial console data buffer before starting the thread to poll the port.

It is expected the spine management controller can receive and act upon all IPMI messages during the node serial console session.

The spine management controller thread for reading of the serial port data should not add artificial delays. The close to real-time serial port output should be immediately cached in the spine management controller serial console buffer and made available for incoming IPMI ReceiveSerialData command.

The management controller thread reading the serial port data should be resilient, and not unexpectedly crash due to error or exception. It is therefore expected once a serial session is established, it will not be unexpectedly lost, including when the node is power cycled.

To ensure stability the management controller must internally maintain a buffer of a minimum 128 kilobytes. Given the default baud-rate of 115200, the buffer would provide resilience from overflow in the worst case condition: 131072 / (115200 / 8) = 9 approx. seconds of buffer time.

All IPMI commands will be encapsulated inside the Send Message payload to direct the spine management controller as to which node (I2C channel) the commands is destined. The StartSerialSession, StopSerialSession, ReceiveSerialData, and SendSerialData commands are the exception. The commands are handled a differently to all other IPMI commands encapsulated in Send Message and forward to the node I2C channel. The StartSerialSession, StopSerialSession, ReceiveSerialData and SendSerialData commands terminate at the spine management controller instead of forwarding the message content, the spine BMC takes the appropriate action on its local serial console buffer for individual nodes. The blade BMC will therefore need to support multiple buffers to support concurrent serial console sessions to different nodes within the blade.

Serial Session Communication overview using a Chassis Manager:



When the Chassis Manager starts a node serial session by sending StartSerialSession request encapsulated in the Send Message Request, the spine management controller will use the channel in the Send Message to determine the target node. The spine management controller will begin polling the target node serial port and buffering the output. The polling of the node serial port is independent of messages received from the Chassis Manager.

The spine management controller will only stop polling the node buffer when it receives a StopSerialSession request encapsulated in the Send Message Request with the target channel of the particular node, or the timeout period expires in the StartSerialSession request.

## Command-Completion Codes

For a list of command completion codes, reference the Intel IPMI Specification 2.0 Section 5.2.

# Spine Command Formats

The following sections describe the formats of the protocol payload commands.

Note that the following sections copy command formats from the IPMI 2.0 specification. In addition to IPMI-defined command formats, this section details system-defined OEM commands and modifications to IPMI command payloads.

## Get Device ID

Intel IPMI Specification section 20.1 describes the **Get Device ID** command.

## Get System GUID

Intel IPMI Specification section 22.14 **Get System GUID** command request.

## Get Channel Authentication Capabilities

The blade should respond to the Get Channel Authentication Capabilities command outside of an active session. The command can also be executed within the context of an active session. See: Intel IPMI Specification section 22.13

## Send Message

The Send Message commands used for bridging IPMI messages between channels and Nodes. For implementation review Intel IPMI Specification section 22.7.

## Master Write-Read

This command can be used for low-level I2C/SMBus write, read, or write-read accesses to private busses behind the management controller. For implementation review Intel IPMI Specification section 22.11

## Chassis Identify

Used to toggle Spine identity LEDs. For implementation review Intel IPMI Specification section 28.5

## Get Sensor Reading Factors

Retrieves Sensor conversion factors. For implementation review Intel IPMI Specification section 35.5

## Get Sensor Reading

Retrieves the reading from a given sensor. For implementation review Intel IPMI Specification section 35.14

## Get Sensor Type

This command is used to retrieve the Sensor Type and Event/Reading Type for the specified sensor. For implementation review Intel IPMI Specification section 35.15

## Read FRU Data

The **Read FRU Data** command returns the specified data from the FRU Inventory Information area. This is a low-level, direct interface to a non-volatile storage area, which means that the interface does not interpret or check any semantics or formatting for the data being accessed. The offset used in this command is a logical offset that might or might not correspond to the physical address used in device that provides the non-volatile storage

## For command implementation review Intel IPMI Specification section 34.2Write FRU Data

The **Write FRU Data** command writes the specified byte or word to the FRU inventory information area. This is a low-level, direct interface to a non-volatile storage area, which means that the interface does not interpret or check any semantics or formatting for the data being accessed. The offset used in this command is a logical offset that might or might not correspond to the physical address used in device that provides the non-volatile storage

## For command implementation review Intel IPMI Specification section 34.3Get SDR

For command implementation review Intel IPMI Specification section 33.12

The Full Sensor Record can be used to describe any type of sensor. The Compact Sensor Record saves space, but is limited in the sensors it can describe. The Full Record is defined as a 64-byte record, while the Compact Record is defined as 48-bytes. Other differences are summarized Table 27

Table . Sensor Record Data Types

| Sensor capability | Supported in full sensor record(type 01h) | Supported in compact sensor record(type 02h) |
| --- | --- | --- |
| Sensor provides analog readings  | yes | no |
| Sensor requires threshold value initialization  | yes | no |
| Sensor requires hysteresis value initialization | yes | yes |
| Multiple sensors can share record  | no | yes |

A threshold-based sensor is a special-case of a discrete sensor. In the **Get Sensor Reading** command, a discrete sensor returns the present status for all discrete states monitored by the sensor, while a threshold-based sensor returns a threshold comparison status.

For this reasons, the Discrete Reading Mask field in the SDR for a discrete sensor is used to indicate which possible states can be read using a **Get Sensor Reading** command, while the Upper and Lower Threshold Reading Mask fields for a threshold-based sensor indicate which thresholds can be read.

## Reserve SEL

The **Reserve SEL** command is used to set the present owner of the SEL, as identified by the software ID (rqAddress) or by the requester’s slave address. The reservation process provides a limited amount of protection when records are being deleted or incrementally read.

A Reservation ID value is returned in response to this command. This value is required in other requests, such as the **Clear SEL** command (commands will not execute unless the correct Reservation ID value is provided).

As an example, if the Chassis Manager wants to clear the SEL, it first reserves the repository by issuing a **Reserve SEL** command. The application checks to see if all SEL entries have been handled before issuing the **Clear SEL** command. If a new event had been placed in the SEL after the records were checked but before the **Clear SEL** command, it is possible for the event to be lost. However, the addition of a new event to the SEL causes the present Reservation ID to be canceled, preventing the **Clear SEL** command from executing. The Chassis Manager can then repeat the reserve-check-clear process until it succeeds.

For command implementation review Intel IPMI Specification section 31.4

## Get SEL Entry

For command implementation review Intel IPMI Specification section 31.5Table 34 shows the system event log record format.

Table . System Event Log Record Format

|  |  |
| --- | --- |
| **Byte** | **Description** |
| 1:2 | Record ID  |
| 3 | Record Type |
| 4:7 | Timestamp |
| 8:9 | Generator Id |
| 10 | Evm Rev |
| 11 | Sensor Type |
| 12 | Sensor # |
| 13 | Event Dir | Event Type |
| 14 | Event Data 1 |
| 15 | Event Data 2 |
| 16 | Event Data 3 |

Note that time is an unsigned 32-bit value representing the local time as the number of seconds from 00:00:00, January 1, 1970 GMT. This format, which is based on a long-standing UNIX-based standard for time keeping, is sufficient to maintain time stamping with 1-second resolution past the year 2100. Similar time formats are used in ANSI C.

## Add SEL Entry

For command implementation review Intel IPMI Specification section 31.6

## Clear SEL

Upon clearing the System Event Log, the Clear Command should append a Log file indicating the date and time the event log was cleared. Clearing the SEL should not block new events from being logged during the clear operations. For command implementation review Intel IPMI Specification section 31.9.

## Get Power Reading

This command should return the spine power reading including the aggregate of all nodes on the spine. This command should read the power consumed by the spine HSC. For command implementation review Intel DCMI 1.5 Specification section 6.6.1

## Get Power Limit

This command should return the blade power reading including the aggregate of all nodes on the spine. For command implementation review Intel DCMI 1.5 Specification section 6.6.2

## Set Power Limit

The command should set an even limit on all nodes in the spine. The limit in watts byte 6:7 should be divided by the number of nodes present on the spine and a power limit applied to each node respectively. For command implementation review Intel DCMI 1.5 Specification section 6.6.3

## Activate Power Limit

This command should be forwarded to all nodes connected to the spine. For command implementation review Intel DCMI 1.5 Specification section 6.6.4

## StartSerialSession

This command starts the BMC polling of the node serial console port. It is expected that this command will start an internal thread that continually polls the serial console port and buffers the data in an internal BMC buffer. The mechanism for buffering console data can also be interrupt driven. It is important that polling is done without delay to ensure that no payload is dropped.

To ensure affinity, the start serial session request command should fail if there is already an active console session.

Multiple buffers will be required to concurrently support all nodes in the blade.

The following tables describe the StartSerialSession request and response.

Table . StartSerialSession Request

|  |  |
| --- | --- |
| Byte | **Description** |
| 1 | [0] Flush Buffer: 1 = Flush Buffer. 0 = Do not flush buffer.[3:1] Node Id[7:4] Inactivity timeout in 30 second increments. 1 –based0h = session does not timeout and close due to inactivity |

Table . StartSerialSession Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | IPMI completion code FDh = Session Already Active.  |
| 2 | 0 = Reserved1 = Session Active/Started |

## ReceiveSerialData

This command is sent from the remote host or Chassis Manager to the spine BMC to receive the contents of the buffer for the node console cache. It is expected that this command will be sent periodically by the Chassis Manager to prevent buffer overflow.

Should a buffer overflow occur in the spine BMC, the BMC should respond with the error code indicating buffer overflow as shown in Table 51. After returning the completion code for buffer overflow once, the spine BMC should automatically clear the buffer and clear the overflow condition and continue filling up the buffer with new data.

Internally the spine BMC must maintain a buffer with a minimum size of 128 kilobytes. Given the baud rate of 115200 this would provide: 131072 / (115200 / 8) = 9 approx. seconds of buffer for the host output.

After every data receive the buffer is freed for new data. Receive requests should not block new data being polled by the BMC from entering the buffer. A typically implementation would accomplish this using double pointers in a circular buffer.

The following tables describe the ReceiveSerialData request and response.

Table . ReceiveSerialData Request

|  |  |
| --- | --- |
| Byte | **Description** |
| 1:2 | BMC serial console buffer length to read. FFFFh implies read entire buffer. The response message should return all occupied bytes in the buffer.  |

Table . ReceiveSerialData Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | IPMI completion code:9A – No Session Active.9B – Buffer Overflow occurred. |
| 2:3 | Payload length (active length) |
| 4:N | Frame encapsulated Serial Console payload from BMC Buffer. |

## SendSerialData

This command is used to send the payload to the node serial console port. The BMC should strip all IPMI framing and message headers then forward the payload to the node console port without any modifications to the payload.

The payload will be limited in size to < 60K. The BMC must therefore reserve adequate buffer for 60K message handling.

The following tables describe the SendSerialData request and response.

Table . SendSerialData Request

|  |  |
| --- | --- |
| Byte | **Description** |
| 1:2 | Payload Length (limited in size to < 60K) |
| 3:N | Payload Data. |

Table . SendSerialData Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | IPMI completion code 9A –No Session Active |

## StopSerialSession

This command stops the serial session. The following tables describe the StopSerialSession request and response.

Table . StopSerialSession Request

|  |  |
| --- | --- |
| Byte | **Description** |
| - |  |

Table . StopSerialSession Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | IPMI completion code  |

## GET GPIO

The **GET GPIO** command asserts in many functions by provide GPIO status.

Table . Get GPIO

| Byte | **Description** |
| --- | --- |
| 1 | Pin Number (zero based) |

Table . Get GPIO Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code00h = Success |
| 2 | Pin Direction (00h for input and 01h for output) |
| 3 | Pin Value (00h for low and 01h for high) |

## SET GPIO

The **SET GPIO** command asserts/deasserts GPIO. This command can be used to validate BMC functionality by driving GPIO positions. Review the baseboard GPIO specification for GPIO functionality. Refer to GPIO specification for pin numbers.

Table . SET GPIO Request

| Byte | **Description** |
| --- | --- |
| 1 | Pin Number (zero based) |
| 2 | Pin Direction (00h for input and 01h for output) |
| 3 | Pin Value (00h for low and 01h for high) |

Table . SET GPIO Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code00h = Success |

## Get CPLD Info

The **Get CPLD Info** command returns the CPLD version information. This command is used to determine the version of the blade motherboard CPLD. This command assists in the support of NVDIMM, whereby different power down sequences may utilized.

Table . Get CPLD Info Request

| Byte | **Description** |
| --- | --- |
| 1 | 0x03 Get CPLD Revision Information (UES code) |

Table . Get CPLD Info Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code: 00h = Success |
| 2:5 | Version Info (LSB first) |

## Write Firmware Update

The **Write Firmware Update** command is used for downloading a new firmware image. The multi node blade firmware is updated over serial. The image is incrementally written to the BMC and stored in flash. During the firmware download the BMC should be available to service any IPMI command request.

Table . Write Firmware Update Request

| Byte | **Description** |
| --- | --- |
| 1:2 | Block Number (block number rolled and not unique) |
| 3:4 | Length |
| 5:N | Payload |

Table . SET GPIO Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code00h = Success |

## Read Firmware Update

The **Read Firmware Update** command is used for reading back the last block written.

Table . SET GPIO Request

| Byte | **Description** |
| --- | --- |
| 1:2 | Block Number to read. |

Table . SET GPIO Response

| Byte | **Description** |
| --- | --- |
| 1:2 | Length |
| 3:N | Payload |

## Firmware Update

The **Firmware Update** command is used to instruct the BMC to load the newly downloaded firmware from flash.

Table . Firmware Update Request

| Byte | **Description** |
| --- | --- |
| 1 | 00 - Return checksum01 - Commit02 - Discard |

Table . Firmware Update Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code00h = Success |
| 2 | Image Checksum |

## Get Nodes Present

The **Get Nodes Present** command is used to instruct the blade BMC to return nodes present in the blade.

Table . Get Node Present

| Byte | **Description** |
| --- | --- |
|  |  |

Table . Get Node Present

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code00h = Success |
| 2 | Present Count |
| 3:N | Node Status: Array of bytes, 1 bye per node:Bit[0] Node Present 0 = No Node Present 1 = Node PresentBit[1] Power State0 = Powered Off1 = Powered OnBit[2-7] Reserved |

## Sensor Data Repository

The Sensor Data Record Formats supported by this specification are: Full Sensor Record (0x01) Compact Sensor Record (0x02) and Event Only (0x03).

SDR Entity’s and entity IDs should adhere to the IPMI 2.0 Specification and uniquely identify a physical entity and instance or logical group of entities and instances within the system:

IPMI 2.0 - 39: “An Entity Id is a standardized numeric code that is used in SDRs to identify the types of physical entities or FRUs in the system. The codes include values for entities such as Processor, Power Supply, Fan, etc. The Entity ID values are specified in IPMI 2.0 Specification Table 43-13, Entity Id Codes.

 The Entity ID is associated with an Entity Instance value that is used to indicate the particular instance of an entity. For example, a system with four processors would use an Entity Instance value of ‘0’ to identify the first processor, ‘1’ for the second, and so on”

Entity Instance values are zero-based. For example, the first temperature sensor will have Entity Instance 0x00.

# System Event Logs

The BMC should support all standard IPMI 2.0 and DCMI system event logs. In addition the following system event log formats should be supported.

#### Hot Swap Controller (HSC) Status

When an event is asserted in the HSC status registers, the BMC will add a SEL entry. The Event Data 1 field of the SEL record indicates the bit offset that generated the event for the following registers:

* HSC Status Byte Low
* HSC Status Byte High
* HSC Status Input

For example, if Event Data 1 = 0x04, that means bit 4 is set to ‘1’ in the relevant register.

For the HSC Status MFR Specific register, the Event Data 1 field indicates the actual register value. For example, if Event Data 1 = 0x2, that means bit 1 is set to ‘1’ in the HSC Status MFR Specific register.

#### Write Firmware Update Log

Firmware update should update using the following SEL format. One SEL entry is logged at the start of the update, and another entry is logged when the update is completed.

OEM Data 2-6 are only used when OEM Data 1[3:0] = 0x0 (Update Started).

Table Firmware Update Event Log

| Byte | **Field** | **Description** |
| --- | --- | --- |
| 1:2 | Record ID  | SEL Record Id. |
| 3 | Record Type | OEM System Event Record: C1h (BIOS Update) |
| 4:7 | Timestamp | Timestamp |
| 8:10 | Manufacturer Id |  |
| 11 | OEM Data 1 | Update Status [7:4]0x0 – Success0xF – FailureUpdate Stage [3:0]0x0 – Update Started0x1 – Update Completed |
| 12 | OEM Data 2 | Current Firmware Version[7:4] Major Version[3:0] Minor Version |
| 13 | OEM Data 3 | Current Firmware Additional Version Information[7:4] Hotfix Version – corresponds to the numeric value in position 6 of the Firmware Naming table[3:0] Development/Test Version – corresponds to position 7 of the Firmware Naming table |
| 14 | OEM Data 4 | New Firmware Version[7:4] Major Version[3:0] Minor Version  |
| 15 | OEM Data 5 | New Firmware Additional Version Information[7:4] Hotfix Version – corresponds to the numeric value in position 6 of the Firmware Naming table[3:0] Development/Test Version – corresponds to position 7 of the Firmware Naming table |
| 16 | OEM Data 6 | Reserved |

# Overview of the Node API

This section details the application programming interface (API) of the SOC Node used in the multi node Cloud Server system.

The node protocol is a small subset of the intelligent platform management interface (IPMI 2.0) commands. IPMI full 2.0 specification is therefore not required for the node communication, only command and I2C message format compatibility with IPMI 2.0 is required. The node communicates to the spine over I2C, the spine management control acts as an I2C proxy for IPMI messages from remote hosts to the node.

Note: For the purpose of completeness this document contains references and abstracts from the IPMI specification: <http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html>

Table 1 lists the IPMI commands that are required or optional for compute nodes and storage nodes. (Note that “M” is mandatory and “O” is optional.) Note: Storage node command requirements are different from compute node commands.

## Manageability Command Set

Table . Required Standard IPMI Commands for Nodes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command name** | **Reference** | **Type** | **Fn** | **Cmd** | **Compute Node** |
| Get Device ID  | 20.1 | App | 06h | 01h | M |
| Get System GUID  | 22.14 | App | 06h | 37h | M |
| Send Message | 22.7 | App | 06h | 34h | M |
| Master Write-Read | 22.11 | App | 06h | 52h | M |
| Get Chassis Status  | 28.2 | Chassis | 00h | 01h | M |
| Chassis Control  | 28.3 | Chassis | 00h | 02h | M |
| Chassis Identify  | 28.5 | Chassis | 00h | 04h | M |
| Set System Boot Options  | 28.12 | Chassis | 00h | 08h | M |
| Get System Boot Options  | 28.13 | Chassis | 00h | 09h | M |
| Get Sensor Reading  | 35.14 | Sensor | 04h | 2Dh | M |
| Get Sensor Type | 35.16 | Sensor | 04h | 2FH | M |
| Get FRU Inventory Area Info | 34.1 | Storage | 0Ah | 10h | M |
| Read FRU Data  | 34.2 | Storage | 0Ah | 11h | M |
| Write FRU Data  | 34.3 | Storage | 0Ah | 12h | M |
| Get SDR  | 33.12 | Storage | 0Ah | 23h | M |
| Reserve SEL  | 31.4 | Storage | 0Ah | 42h | M |
| Get SEL Info | 31.2 | Storage | 0Ah | 40h | M |
| Get SEL Entry  | 31.5 | Storage | 0Ah | 43h | M |
| Add SEL Entry | 31.6 | Storage | 0Ah | 44h | M |
| Clear SEL  | 31.9 | Storage | 0Ah | 47h | M |
| Get Power Reading | N/A | DCMI | 2Ch | 02h | M |
| Get Power Limit | N/A | DCMI | 2Ch | 03h | M |
| Set Power Limit | N/A | DCMI | 2Ch | 04h | M |
| Activate Power Limit | N/A | DCMI | 2Ch | 05h | M |

*M**= Mandatory*

Table . Required OEM IPMI Commands for Nodes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command name** | **Reference** | **Type** | **Fn** | **Cmd** | **Compute Node** |
| Send request message to BMC | N/A | OEM | 38h | 01h | M |
| Get all GPIO status | N/A | OEM | 38h | 03h | M |
| Set all GPIO status | N/A | OEM | 38h | 04h | M |
| Get GPIO configuration | N/A | OEM | 38h | 05h | M |
| Set GPIO configuration | N/A | OEM | 38h | 06h | M |
| Send interrupt to BMC | N/A | OEM | 38h | 07h | M |
| Get BIOS Code  | N/A | OEM | 38h | 08h | M |
| Firmware Update  | N/A | OEM | 38h | 09h | M |
| Firmware Verify | N/A | OEM | 38h | 0Ah | M |
| Get Firmware Version  | N/A | OEM | 38h | 0Bh | M |

# Node Implementation Requirements

The following sections describe the blade implementation requirements.

Only a small subset of commands of IPMI2.0 is required for node support.

## Interfaces

* I2C

## IPMI Bus Support

* System Bus (system Interface)
* SMBus (I2C)
* IPMB

## System Event Log

The System Event Log (SEL) should implement circular logging. The maximum number of records in the circular log should be 226 records or roughly 4KB in memory. When the SEL is full, the log should overwrite the oldest record with the newest record.

## SMBus Slave Address

The node BMC must have the SMbus Slave Address must be configurable and have the default SMbus address of 0x40(8-bit) for the node-bmc and and spine-bmc is 0x20(8-bit)

## GPIO Mapping

GPIO pin mappings are listed in the Node GPIO appendix. This document will detail the BMC pin assignments.

## Keyboard Controller Style (KCS) Interface

The node should support in-band (host to management controller) IPMI communication.

## Firmware Decompression

The node management controller firmware must deterministically be fully decompressed (loaded) and ready to service IPMI request messages within 20 seconds from the time when DC power is applied to the baseboard.

## Inlet Sensor

The blade Inlet sensor should be always be sensor number B0h in the Sensor Data Repository.

## **Serial Console Support**

The node management controller should support control logic enables it to monitor the console redirection port.

## **Node Serial Console Support**

During normal operation, the console output is streaming uninterrupted from the node. The node-BMC is not monitoring or encapsulating the payload.

The spine-BMC is perpetually receiving serial console output from the node UART.

## Node Identification Commands

The **Get Channel Authentication Capabilities** command is used to identify the node type. A node should always respond to this command. The response message should use byte 9 (OEM auxiliary data). When response message byte 9 is combined with the OEM ID (bytes 6:8), the spine-BMC will know both the manufacturer and type of node.

A node should zero byte 3 and byte 4. As authentication over SMBUS is not supported. All mandatory (M) IPMI commands listed in this specification should be supported without a session initialization processes or authentication mechanism.

# Node Command Formats

The following sections describe the node command formats for the supported commands.

## Get Device ID

For implementation information refer to the IPMI 2.0 specification section 20.1

## Get System GUID

Refer to the IPMI 2.0 specification section 22.14

## Send Message

The Send Message commands used for bridging IPMI messages between channels, and between the system management software (SMS) and a given channel. Refer to the IPMI 2.0 specification section 22.7

## Master Write-Read

This command can be used for low-level I2C/SMBus write, read, or write-read accesses to private busses behind the management controller. Refer to the IPMI 2.0 specification section 22.11

## Get Chassis Status

**Refer to the IPMI 2.0 specification section 28.2**

## Chassis Control

Refer to the IPMI 2.0 specification section 28.3.For the power cycle command, if the node is currently powered off, the blade should be turned on.

## Chassis Identify

Changes maintenance LED indicator on the node. Refer to the IPMI 2.0 specification section 28.5

## Set Power Restore Policy

The **default** value in the node-BMC should be to always power up after AC/mains is applied. Refer to the IPMI 2.0 specification section 28.8

## Set System Boot Options

Refer to the IPMI 2.0 specification section 28.12

## Get System Boot Options

Refer to the IPMI 2.0 specification section 28.13

## Get Sensor Reading

Refer to the IPMI 2.0 specification section 35.14

## Get Sensor Type

This command is used to retrieve the Sensor Type and Event/Reading Type for the specified sensor. Refer to the IPMI 2.0 specification section 25.15

## Get FRU Inventory Area Info

Refer to the IPMI 2.0 specification section 34.1

## Read FRU Data

The **Read FRU Data** command returns the specified data from the FRU Inventory Information area. This is a low-level, direct interface to a non-volatile storage area, which means that the interface does not interpret or check any semantics or formatting for the data being accessed. The offset used in this command is a logical offset that might or might not correspond to the physical address used in device that provides the non-volatile storage

Refer to the IPMI 2.0 specification section 34.2

## Write FRU Data

The **Write FRU Data** command writes the specified byte or word to the FRU inventory information area. This is a low-level, direct interface to a non-volatile storage area, which means that the interface does not interpret or check any semantics or formatting for the data being accessed. The offset used in this command is a logical offset that might or might not correspond to the physical address used in device that provides the non-volatile storage

Refer to the IPMI 2.0 specification section 34.3

## Reserve SDR Repository

The reservation process provides a limited amount of protection on repository access when records are being incrementally read. To understand this command, it is important to understand the process for requesting the SDR:

**Get SDR Command**The **Get SDR** command returns the sensor record specified by “Record ID.” The command also accepts a byte range specification that allows just a selected portion of the record to be retrieved (incremental read). The requester must first reserve the SDR Repository using the **Reserve SDR Repository** command for an incremental read to an offset other than 0000h to be accepted.

If Record ID is specified as 0000h, this command returns the Record Header for the first SDR in the repository. FFFFh specifies that the last SDR in the repository should be listed. If Record ID is non-zero, the command returns the information from the matching record and the Record ID for the next SDR in the repository. When the Chassis Manager wishes to retrieve the full set of SDR records, it must first issue the **Get SDR** command, starting with 0000h as the Record ID to get the first record. The next Record ID is extracted from the response, and this is then used as the Record ID in a **Get SDR** request for the next record. This is repeated until the Last Record ID value (FFFFh) is returned in the Next Record ID field of the response.

**Implementation**If the BMC can buffer and send a response to get SDR in excess of 67 bytes, the **Reserve SDR** command is not required to incrementally read the SDR. In this case, the Get SDR Bytes to Read would be FFh, which means “read entire record.”

Refer to the IPMI 2.0 specification section 33.11

## Get SDR

Refer to the IPMI 2.0 specification section 33.12The Full Sensor Record can be used to describe any type of sensor. The Compact Sensor Record saves space, but is limited in the sensors it can describe. The Full Record is defined as a 64-byte record, while the Compact Record is defined as 48-bytes. Other differences are summarized Table 27

Table . Sensor Record Data Types

| Sensor capability | Supported in full sensor record(type 01h) | Supported in compact sensor record(type 02h) |
| --- | --- | --- |
| Sensor provides analog readings  | yes | no |
| Sensor requires threshold value initialization  | yes | no |
| Sensor requires hysteresis value initialization | yes | yes |
| Multiple sensors can share record  | no | yes |

A threshold-based sensor is a special-case of a discrete sensor. In the **Get Sensor Reading** command, a discrete sensor returns the present status for all discrete states monitored by the sensor, while a threshold-based sensor returns a threshold comparison status.

For this reasons, the Discrete Reading Mask field in the SDR for a discrete sensor is used to indicate which possible states can be read using a **Get Sensor Reading** command, while the Upper and Lower Threshold Reading Mask fields for a threshold-based sensor indicate which thresholds can be read.

##  Reserve SEL

The **Reserve SEL** command is used to set the present owner of the SEL, as identified by the software ID (rqAddress) or by the requester’s slave address. The reservation process provides a limited amount of protection when records are being deleted or incrementally read.

A Reservation ID value is returned in response to this command. This value is required in other requests, such as the **Clear SEL** command (commands will not execute unless the correct Reservation ID value is provided).

As an example, if the Chassis Manager wants to clear the SEL, it first reserves the repository by issuing a **Reserve SEL** command. The application checks to see if all SEL entries have been handled before issuing the **Clear SEL** command. If a new event had been placed in the SEL after the records were checked but before the **Clear SEL** command, the events should not be cleared.

Refer to the IPMI 2.0 specification section 31.4

## Get SEL Info

Refer to the IPMI 2.0 specification section 31.2

## Get SEL Entry

Refer to the IPMI 2.0 specification section 31.5Table 117 shows the system event log record format.

Table . System Event Log Record Format

|  |  |
| --- | --- |
| **Byte** | **Description** |
| 1:2 | Record ID  |
| 3 | Record Type |
| 4:7 | Timestamp |
| 8:9 | Generator Id |
| 10 | Evm Rev |
| 11 | Sensor Type |
| 12 | Sensor # |
| 13 | Event Dir | Event Type |
| 14 | Event Data 1 |
| 15 | Event Data 2 |
| 16 | Event Data 3 |

Table 118 shows a sample system event log response message with two events in the log.

Table . System Event Log Record Format

|  |  |  |
| --- | --- | --- |
| **Byte** | **Payload** | **Data field** |
| 0 | 0x00 | Completion Code |
| 1 | 0x16 | LS Byte Count |
| 2 | 0x00 | LS Byte Count |
| 3 | 0x01 | Record Id LS |
| 4 | 0x00 | Record Id MS |
| 5 | 0x00 | Record Type |
| 7 | 0xED | Timestamp seconds LS - MS from January 1, 1970 |
| 8 | 0x43 | Timestamp seconds LS - MS from January 1, 1970 |
| 9 | 0x73 | Timestamp seconds LS - MS from January 1, 1970 |
| 10 | 0x4F | Timestamp seconds LS - MS from January 1, 1970 |
| 11 | 0x00 | Generator Id |
| 12 | 0x00 | Generator Id |
| 13 | 0x01 | Sensor Type |
| 14 | 0x03 | Sensor Number 3 |
| 15 | 0x02 | Event Dir | Event Type |
| 16 | 0x60 | Event Data 1 |
| 17 | 0x00 | Event Data 2 |
| 18 | 0xFF | Event Data 3 |
| 19 | 0x01 | Record Id LS |
| 20 | 0x00 | Record Id MS |
| 21 | 0x00 | Record Type |
| 22 | 0xED | Timestamp seconds LS - MS from January 1, 1970 |
| 23 | 0x43 | Timestamp seconds LS - MS from January 1, 1970 |
| 24 | 0x73 | Timestamp seconds LS - MS from January 1, 1970 |
| 25 | 0x4F | Timestamp seconds LS - MS from January 1, 1970 |
| 26 | 0x00 | Generator Id |
| 27 | 0x00 | Generator Id |
| 28 | 0x07 | SensorType Processor |
| 29 | 0x01 | Sensor Number 1 |
| 30 | 0x41 | Desertion Event / Type Code 0x01 |
| 31 | 0x51 | Event Data 1 |
| 32 | 0x00 | Event Data 2 |
| 33 | 0xFF | Event Data 3 |

Note that time is an unsigned 32-bit value representing the local time as the number of seconds from 00:00:00, January 1, 1970 GMT. This format, which is based on a long-standing UNIX-based standard for time keeping, is sufficient to maintain time stamping with 1-second resolution past the year 2100. Similar time formats are used in ANSI C.

## Add SEL Entry

Refer to the IPMI 2.0 specification section 31.6

## Clear SEL

Upon clearing the System Event Log, the Clear Command should append a Log file indicating the date and time the event log was cleared.

Refer to the IPMI 2.0 specification section 31.9

## Get Power Reading

For command implementation review Intel DCMI 1.5 Specification section 6.6.1

## Get Power Limit

For command implementation review Intel DCMI 1.5 Specification section 6.6.2

## Set Power Limit

. For command implementation review Intel DCMI 1.5 Specification section 6.6.3

## Activate Power Limit

For command implementation review Intel DCMI 1.5 Specification section 6.6.4

## Send Request Message to BMC

Send Request Message to BMC Request describes the **Send Request Message to BMC** request.

Table . Send Request Message to BMC Request

| Byte | **Description** |
| --- | --- |
| 1 | Request InterfaceO1h Mgmt Ctrl02h Serial03h KCS  |
| 2:N | Request Data |

Table 132. Send Request Message to BMC Response describes the **Send Request Message to BMC** response.

Table . Send Request Message to BMC Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | Completion code |
| 2 | Request InterfaceO1h Mgmt Ctrl02h Serial03h KCS |
| 3:N | Response Data |

## Get All GPIO Status

Table 133. Get All GPIO Status Request describes the Get All GPIO Status request. This command used by the spine BMC to get GPIO status from the node. Refer to GPIO specification for GPIO mappings.

Table . Get All GPIO Status Request

| Byte | **Description** |
| --- | --- |
|  |  |

Table 134. Get All GPIO Status Response describes the **Get All GPIO Status** response.

Table . Get All GPIO Status Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | Completion code |
| 2:5 | All GPIO status0b low1b High |

## Set All GPIO Status

Table 133. Get All GPIO Status Request describes the Set All GPIO Status request. This command used by the spine BMC to set GPIO status on the node. Refer to GPIO specification for GPIO mappings.

Table . Set All GPIO Status Request

| Byte | **Description** |
| --- | --- |
| 1:3 | GPIO enable Mask0b: Disable1b: Enable |
| 4:7 | Set all GPIO status0b: Low1b: High |

Table 136. Set All GPIO Status Response describes the **Set All GPIO Status** response.

Table . Set All GPIO Status Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | Completion code |

## Get GPIO Configuration

Table 137 Table 133. Get All GPIO Status Request describes the Set All GPIO Status request. This command used by the spine BMC to get the configuration of the node. Refer to GPIO specification for GPIO mappings.

Table . Get GPIO Configuration Request

| Byte | **Description** |
| --- | --- |
| 1:3 | GPIO Enable Mask0b: Disable1b: Enable |

Table 138 describes the **Get GPIO Configuration** response.

Table . Get GPIO Configuration Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | Completion code |
| 2:N | GPIO Configuration (1 byte per GPIO pin)Bit[0] Input/Output0b: Input1b: OutputBit[1] Interrupt disable/enable0b: Disable1b: EnableBit[2] Edge trigger0b: Edge trigger(default)Bit[3:4] Trigger Type00b: Falling edge01b: Rising edge10b: both11b: Reserved |

## Set GPIO Configuration

Table 133. Get All GPIO Status Request describes the Set All GPIO Status request. This command used by the spine BMC to get the configuration of the node. Refer to GPIO specification for GPIO mappings.

Table . Set GPIO Configuration Request

| Byte | **Description** |
| --- | --- |
| 1:3 | GPIO Enable Mask0b: Disable1b: Enable |
| 4:N | GPIO Configuration (1 byte per GPIO pin)Bit[0] Input/Output0b: Input1b: OutputBit[1] Interrupt disable/enable0b: Disable1b: EnableBit[2] Edge trigger0b: Edge trigger(default)Bit[3:4] Trigger Type00b: Falling edge01b: Rising edge10b: both11b: Reserved |

Table 136. Set All GPIO Status Response describes the **Set GPIO Configuration** response.

Table . Set GPIO Configuration Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | Completion code |

## Send Interrupt to BMC

Table 133. Get All GPIO Status Request describes the Set All GPIO Status request. This command used for interrupt notification from the node BMC to the spine BMC. Refer to GPIO specification for GPIO mappings.

Table . Send Interrupt to BMC

| Byte | **Description** |
| --- | --- |
| 1 | Interrupt GPIO Number (Refer to GPIO specification) |
| 2 | Interrupt Trigger Type00b: Falling edge01b: Rising edge |

Table 136. Set All GPIO Status Response describes the **Send Interrupt to BMC** response.

Table . Send Interrupt to BMC Response

|  |  |
| --- | --- |
| Bytes | Description |
| 1 | Completion code |

## Get BIOS Code

The **Get BIOS Code** command returns the BIOS post code history. The history will be stored in memory allocated by the SNOOP driver. It will assign a total of 512 bytes for the current and previous post code. When the system is reset the driver starts to record the current BIOS post code.

Table . Get BIOS Code Request

| Byte | **Description** |
| --- | --- |
| 1 | Command:0 = Read Current BIOS Code1 = Read Previous BIOS code |

Table . Get BIOS Code Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code00h = SuccessCCh = Invalid Data FieldC1h = Invalid Command07h=File Error |
| 2:257 | BIOS CodeByte array of BIOS codes in hexadecimal |

## Firmware Update

The **Firmware Update** command is used to update the BIOS and CLPD from the BMC.

Table . Firmware Update Request

| Byte | **Description** |
| --- | --- |
| 1 | Update Target:00h: BIOS01h: CPLD |
| 2:5 | Offset |
| 6:7 | Data Length |
| 8:N | Image Data |

Table . Firmware Update Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code80h –Write flash error81h – Power status check fail82h – Data length error83h – Flash erase error |

## Firmware Verify

The **Firmware Verify** command is used to update the BIOS and CLPD from the BMC.

Table . Firmware Update Request

| Byte | **Description** |
| --- | --- |
| 1 | Update Target:00h: BIOS01h: CPLD |
| 2:5 | Offset |
| 6:7 | Data Length |
| 8:N | Image Data |

Table . Firmware Update Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code80h – Checksum error82h – Data length error84h – Read flash error |

## Get Firmware Version

The **Get Firmware Version** command is used to get the BIOS, CLPD, BMC or Management Controller Firmware version.

Table . Get Firmware Version Request

| Byte | **Description** |
| --- | --- |
| 1 | Firmware Target:00h: BIOS01h: CPLD02h: BMC03h: Management Controller |

Table . Get Firmware Version Response

| Byte | **Description** |
| --- | --- |
| 1 | Completion Code |
| 2:10 | Firmware Version. ASCII encoded |

# Sensor Data Repository

The first record in the Sensor Data Repository should support Full Sensor Record (0x01) Compact Sensor Record (0x02) and Event Only (0x03).

SDR Entity’s and entity IDs should adhere to the IPMI 2.0 Specification and uniquely identify a physical entity and instance or logical group of entities and instances within the system:

IPMI 2.0 - 39: “An Entity Id is a standardized numeric code that is used in SDRs to identify the types of physical entities or FRUs in the system. The codes include values for entities such as Processor, Power Supply, Fan, etc. The Entity ID values are specified in IPMI 2.0 Specification Table 43-13, Entity Id Codes.

 The Entity ID is associated with an Entity Instance value that is used to indicate the particular instance of an entity. For example, a system with four processors would use an Entity Instance value of ‘0’ to identify the first processor, ‘1’ for the second, and so on”

Entity Instance values are zero-based. For example, the first temperature sensor will have Entity Instance 0x00.

# Appendix 1: Reference Material

This section provides reference material.

## Sensor Types

Refer to Intel IPMI 2.0 Specification Section: 42.2

## Event Reading Codes

Refer to Intel IPMI 2.0 Specification Section: 42.1

## Sensor Unit Type Codes

Refer to Intel IPMI 2.0 Specification Section: 43.17

## Entity IDs

Refer to Intel IPMI 2.0 Specification Section: 43.14

# System Event Logs

The BMC should support all standard IPMI 2.0 and DCMI system event logs. In addition the following system event log formats should be supported.

## System Event Logs Generated by BMC

This section describes the System Event Logs that are generated by the node-BMC.

## System Event Logs Generated by BIOS

This section describes the System Event Logs that are generated by the BIOS.

### Memory ECC Error Logging

The BIOS logs memory ECC errors using the following SEL format..

Table Memory ECC Error Event Log

| Byte | **Field** | **Description** |
| --- | --- | --- |
| 1:2 | Record ID  | SEL Record Id. |
| 3 | Record Type | System Event Record: 02h |
| 4:7 | Timestamp | Timestamp |
| 8:9 | Generator Id | Byte 1 = 01h (Generated by BIOS)Byte 2 = 00h |
| 10 | Evm Rev | 04h |
| 11 | Sensor Type | 0Ch |
| 12 | Sensor # | 87h |
| 13 | Event Dir | Event Type | [7] Assertion: 0[6:0] Event Trigger: 6Fh |
| 14 | Event Data 1 | Bit [7:6] 10b OEM code in byte 2Bit [5:4] 10b OEM code in byte 3Bit [3:0] Offset from Event/Reading Code for discrete event status00b = Correctable Error01b = Uncorrectable Error05b = Correctable ECC error logging Limit Reached [limit TBD] |
| 15 | Event Data 2 | 00h = Single Bit Error warning threshold (Event/Reading Type Code = 0h for Correctable Error) if supported.01h = Single Bit Error critical threshold (Event/Reading Type Code = 5h for Correctable ECC error logging limit reached) if supported.0FFh = unspecifiedOther values are reserved |
| 16 | Event Data 3 | DIMM Number (1-base) |

###  PCI Express Error Logging

The BIOS logs PCIe errors using the following SEL format. The log entries should be added to the SEL in the order shown here.

Table PCIe Error Event Log 1

| Byte | **Field** | **Description** |
| --- | --- | --- |
| 1:2 | Record ID  | SEL Record Id. |
| 3 | Record Type | System Event Record: 02h |
| 4:7 | Timestamp | Timestamp |
| 8:9 | Generator Id | Byte 1 = 01h (Generated by BIOS)Byte 2 = 00h |
| 10 | Evm Rev | 04h |
| 11 | Sensor Type | 13h (Critical Interrupt) |
| 12 | Sensor # | A1h |
| 13 | Event Dir | Event Type | [7] Assertion: 0[6:0] Event Trigger: 6Fh |
| 14 | Event Data 1 | Bit [7:6] 10b Bit [5:4] 10bBit [3:0] 07h = Bus Correctable Error08h = Bus Uncorrectable Error0Ah = Bus Fatal Error |
| 15 | Event Data 2 | Bit [7:3] Device NumberBit [2:0] Function Number |
| 16 | Event Data 3 | Bit [7:0] Bus Number |

Table PCIe Error Event Log 2

| Byte | **Field** | **Description** |
| --- | --- | --- |
| 1:2 | Record ID  | SEL Record Id. |
| 3 | Record Type | System Event Record: 02h |
| 4:7 | Timestamp | Timestamp |
| 8:9 | Generator Id | Byte 1 = 01h (Generated by BIOS)Byte 2 = 00h |
| 10 | Evm Rev | 04h |
| 11 | Sensor Type | 13h (Critical Interrupt) |
| 12 | Sensor # | A1h |
| 13 | Event Dir | Event Type | [7] Assertion: 0[6:0] Event Trigger: 6Fh |
| 14 | Event Data 1 | Bit [7:6] 10b Bit [5:4] 10bBit [3:0] 07h = Bus Correctable Error08h = Bus Uncorrectable Error0Ah = Bus Fatal Error |
| 15 | Event Data 2 | 1st Error ID (see following table for details) |
| 16 | Event Data 3 | 2nd Error ID (see following table for details) |

Table PCIe Error Event Log 2 Error ID Detail

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | **Error** | **Default Error Severity** | **Transaction Response** | **Default Error Logging** |
| 70 | Receiver Error | 0 | Respond per PCI Express specification | CORERRSTS |
| 71 | Bad TLP  | 0 | Respond per PCI Express specification  | CORERRSTS  |
| 72 | Bad DLLP  | 0 | Respond per PCI Express specification  | CORERRSTS  |
| 73 | Replay Time-out  | 0 | Respond per PCI Express specification  | CORERRSTS  |
| 74 | Replay Number Rollover  | 0 | Respond per PCI Express specification  | CORERRSTS  |
| 75 | Received ERR\_COR message from downstream device  | 0 | Respond per PCI Express specification  | RPERRSTS  |
| 76 | PCI Express Link Bandwidth changed  | 0 | No Response -- This error is not associated with a cycle. I/O module detects and logs the error. Log per "Link bandwidth change notification mechanism" ECN  | XPCORERRSTS  |
| 78 | Advisory Non-fatal Error  | 0 | Respond per PCI Express specification  | CORERRSTS  |
| 80 | Received "Unsupported Request" completion status from downstream device | 1 | Coherency interface to PCI Express read: I/O module returns all 1s' and normal response to the coherent interface to indicate master abort Coherency interface to PCI Express NP write: I/O module returns normal response PCI Express to PCI Express read/NP-write: 'Unsupported request' is returned2 to original PCI Express requester.  | XPUNCERRSTS  |
| 81 | Sent a PCI Express "Unsupported Request" response, on inbound request for address decode, request type, or other reason  | 1 | PCI Express read: "Unsupported request" completion is returned on PCI Express PCI Express non-posted write: 'Unsupported request' completion is returned on PCI Express. The write data is dropped PCI Express posted write: I/O module drops the write data. Header is logged where possible.  | XPUNCERRSTS HDRLOG  |
| 82 | Received "Completer Abort" completion status from downstream device  | 1 | Coherency interface to PCI Express read: I/O module returns all '1s' and normal response to the coherency interface Coherency interface to PCI Express NP write: I/O module returns normal response PCI Express to PCI Express read/NP-write: Completer Abort’ is returned 3 to original PCI Express requester.  | XPUNCERRSTS  |
| 83 | Sent a PCI Express "Completer Abort" condition on inbound request for address decode, request type, or other reason  | 1 | PCI Express read: 'Completer Abort' completion is returned on PCI Express PCI Express non-posted write: 'Completer Abort' completion is returned on PCI Express. The write data is dropped PCI Express posted write: I/O module drops the write data. Header is logged where possible.  | XPUNCERRSTS HDRLOG  |
| 84 | Completion timeout on NP transactions outstanding on PCI Express/DMI  | 1 | Coherency interface to PCI Express read: I/O module returns normal response to the coherency interface and all 1’s for read data Coherency interface to PCI Express nonposted write: I/O module returns normal response to the coherency interface PCI Express to PCI Express read/nonposted write: UR2 is returned on PCI Express Header is logged where possible.  | UNCERRSTS HDRLOG  |
| 85 | Received PCI Express Poisoned TLP  | 1 | Outbound-Read Completion or Inbound Write: the packet is sent to the coherent interface. If POISFEN bit is set, the poison packet is set to its destination with poison bit set. If POISFEN is cleared, the packet is dropped. Peer to Peer read completion or write request: I/O module forwards packet with poisoned data to the destination port normally, but with the poison bit set. If POISFEN bit is set and the poison TLP severity is set as non-fatal, then this error is logged as an Advisory Non-fatal. Also, received poisoned TLPs that are not forwarded over the coherency interface are always treated as an Advisory Nonfatal error, if severity is set to non-fatal. Header is logged where possible.  | UNCERRSTS HDRLOG  |
| 86 | Received PCI Express unexpected Completion  | 1 | Respond Per PCI Express Specification Header is logged where possible.  | UNCERRSTS HDRLOG  |
| 87 | PCI Express Flow Control Protocol Error4  | 1 | Respond Per PCI Express Specification  | UNCERRSTS  |
| 88 | Received ERR\_NONFATAL Message from downstream device  | 1 | Respond per PCI Express specification  | RPERRSTS  |
| 89 | Received a Request from a downstream component that is unsupported  | 1 | For Non-posted requests: 'unsupported request' response is sent and a bit in XPUNCERRSTS is logged. For posted requests, they are simply logged and dropped. Header is logged where possible.  | UNCERRSTS HDRLOG  |
| 8A | Received a Request from a downstream component that is to be completer aborted  | 1 | For Non-posted requests: 'completer abort' response is sent and a bit in XPUNCERRSTS is logged. For posted requests, they are simply logged and dropped. Header is logged where possible.  | UNCERRSTS HDRLOG  |
| 8B | ACS Violation  | 1 | Respond per PCI Express specification. An ACS violation will cause a violating request to be aborted and logged. Header is logged where possible.  | UNCERRSTS HDRLOG  |
| 90 | PCI Express Malformed TLP4  | 2 | Respond Per PCI Express Specification Header is logged where possible.  | UNCERRSTS HDRLOG  |
| 91 | PCI Express Data Link Protocol Error4  | 2 | Respond Per PCI Express Specification  | UNCERRSTS  |
| 92 | PCI Express Receiver Overflow  | 2 | Respond Per PCI Express Specification  | UNCERRSTS  |
| 93 | Surprise Down  | 2 | Respond Per PCI Express Specification  | UNCERRSTS  |
| 94 | Received ERR\_FATAL message from downstream device  | 2 | Respond Per PCI Express Specification  | RPERRSTS  |
| 97 | Outbound switch header queue parity error  | 2 | Undefined. This is a fatal error for the given root port.  | XPUNCERRSTS  |
| 98 | MSI writes greater than a DWORD  | 1 | Drop the transaction  | XPUNCERRSTS  |
| 99 | Outbound Poisoned Data  | 1 | The poisoned data packet is sent, but logged here.  | XPUNCERRSTS  |

Table PCIe Error Event Log 3

| Byte | **Field** | **Description** |
| --- | --- | --- |
| 1:2 | Record ID  | SEL Record Id. |
| 3 | Record Type | OEM System Event Record: C0h |
| 4:7 | Timestamp | Timestamp |
| 8:10 | Manufacturer Id |  |
| 11:12 | Vendor ID | Vendor ID of error source device |
| 13:14 | Device ID | Device ID of error source device |
| 15 | OEM Data 1 | FFh |
| 16 | OEM Data 2 | FFh |

# Appendix: Commonly Used of Acronyms

This section provides definitions of acronyms used in this specifications.

**ACPI –** advanced configuration and power interface

**AHCI** – advanced host controller interface

**AHJ** – authority having jurisdiction

**ANSI** – American National Standards Institute

**API** – application programming interface

**ASHRAE** – American Society of Heating, Refrigerating and Air Conditioning Engineers

**ASIC** – application-specific integrated circuit

**BCD** – binary-coded decimal

**BIOS** – basic input/output system

**BMC** – baseboard management controller

**CFM** – cubic feet per minute (measure of volume flow rate)

**CM** – Chassis Manager

**CMOS** – complementary metal–oxide–semiconductor

**COLO** – co-location

**CTS** – clear to send

**DCMI –** Data Center Manageability Interface

**DDR3** – double data rate type 3

**DHCP** – dynamic host configuration protocol

**DIMM** – dual inline memory module

**DPC** – DIMMs per memory channel

**DRAM** – Dynamic Random Access Memory

**DSR** – data set ready

**DTR** – data terminal ready

**ECC** – error-correcting code

**EEPROM** - electrically erasable programmable read-only memory

**EIA** – Electronic Industries Alliance

**EMC** – electromagnetic compatibility

**EMI** – electromagnetic interference

**FRU** – field replaceable unit

**FTP** – file transfer protocol

**GPIO** – general purpose input output

**GUID**  – globally unique identifier

**HBI** – high business intelligence

**HCK** – Windows Hardware Certification Kit

ard

**HMD** – hardware monitoring device

**HT** – hyperthreading

**I2C** – inter-integrated circuit

**IBC** – international building code

**IDE** – integrated development environment

**IEC** - International Electrotechnical Commission

**IOC** – I/O controller

**IPMI** – intelligent platform management interface

**IPsec** – IP security

**ITPAC** – IT pre-assembled components

**JBOD** – “just a bunch of disks”

**KCS** – keyboard controller style

**L2** – layer 2

**LAN** – local area network

**LFF –** large form factor

**LPC** – low pin count

**LS** – least significant

**LUN** – logical unit number

**MAC** – media access control

**MDC** – modular data center containers

**MLC** – multi-level call

**MTBF** – mean time between failures

**MUX** - multiplexer

**NIC** – network interface card

**NUMA** – non-uniform memory access

**OOB** – out of band

**OSHA -** Occupational Safety & Health Administration

**OTS** – off the shelf

**PCB** – printed circuit board

**PCIe** – peripheral component interconnect express

**PCH** – platform control hub

**PDB** – power distribution backplane

**PDU** – power distribution unit

**Ph-ph** – phase to phase

**Ph-N** – phase to neutral

**PNP** – plug and play

**POST** – power-on self-test

**PSU** – power supply unit

**PWM** – pulse-width modulation

**PXE** – preboot execution environment

**QDR** – quad data rate

**QFN** – quad flat package no-lead

**QPI** – Intel QuickPath Interconnect

**QSFP** – Quad small form-factor pluggable

**RAID** – redundant array of independent disks

**REST -** representational state transfer

**RM** – Rack Manager

**Rma** – remote management agent

**ROC** – RAID-on-chip controller

**RSS** – receive-side scaling

**RTS** – request to send

**RU** – rack unit

**RxD** – received data

**SAS** – serial-attached small computer system interface (SCSI)

**SATA** – serial AT attachment

**SCK** – serial clock

**SCSI** – small computer system interface

**SDA** – serial data signal

**SDR** – sensor data record

**SFF** – small form factor

**SFP** - small form-factor pluggable

**SMBUS** – systems management bus

**SMBIOS** – systems management BIOS

**SOL** – serial over LAN

**SPI** – serial peripheral interface

**SSD** – solid-state drive

**TB** – tray backplane

**TDP** – thermal design power

**TM** – tray midplane

**TOR** – top of rack

**TPM** – trusted platform module

**TxD** – transmit data

**U** – rack unit

**UART** – universal asynchronous receiver/transmitter

**UEFI** – unified extensible firmware interface

**UL** – Underwriters Laboratories

**UPS** – uninterrupted power supply

**Vpp** – voltage peak to peak

**WMI** –Windows Management Interface