

## Application Guide ACAN-05

### cPCI325DC IPMI Satellite Controller Specification

On the pages that follow, you will find the preliminary specification for the IPMI/IPMB Satellite Controller used in cPCI325DC CompactPCI power supplies. The firmware was developed by Philips Semiconductors using their P89C662HBA/00 controller. Detailed information on this controller, including features, complete specifications, application notes, parametrics, support and tools, can be found at <http://www.semiconductors.philips.com/pip/p89c662hba/00>

# Philips Semiconductors

Preliminary Specification

IPMI/IPMB Satellite Controller for Power Supply Applications

James P. Earle  
Market Applications Engineer  
Philips Semiconductors  
2140 Lake Park Blvd., Suite 200  
Richardson, Texas 75230  
972-705-2485  
[jim.earle@philips.com](mailto:jim.earle@philips.com)

## **Table of Contents**

Section 1	Introduction to IPMI/IPMB
Section 2	Implementation
Section 3	Application Commands
Section 4	Sensor/Event Commands
Section 5	Firmware Commands
Section 6	Storage Commands - FRU Device
Section 7	OEM Commands
Section 8	Error Code Support

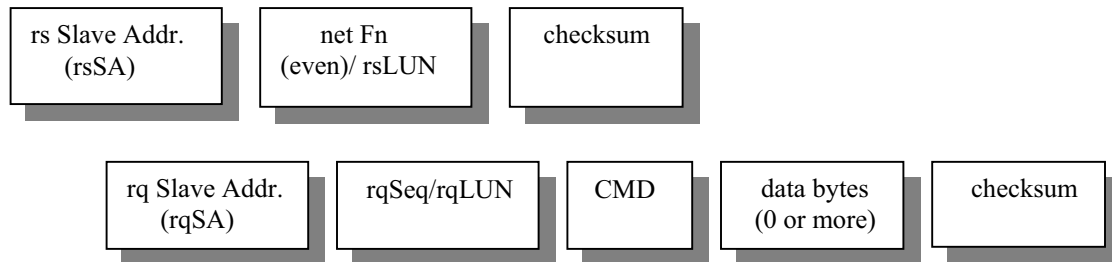
## **Appendices**

Appendix A	Get Device ID Detail
Appendix B	Sensor Data Record (SDR) Detail
Appendix C	Table C-1, Event/Reading Type Code Ranges (Table 36-1) Table C-2, Generic Event/Reading Type Codes (Table 36-2) Table C-3, Sensor Type Codes (Table 36-3)
Appendix D	Entity ID's (Table 37-12)
Appendix E	Sensor Unit Type Codes (Table 37-14)
Appendix F	Sensor Reading Conversion Formula
Appendix G	FRU Inventory Formats
Appendix H	IPMI/IPMB Reference List

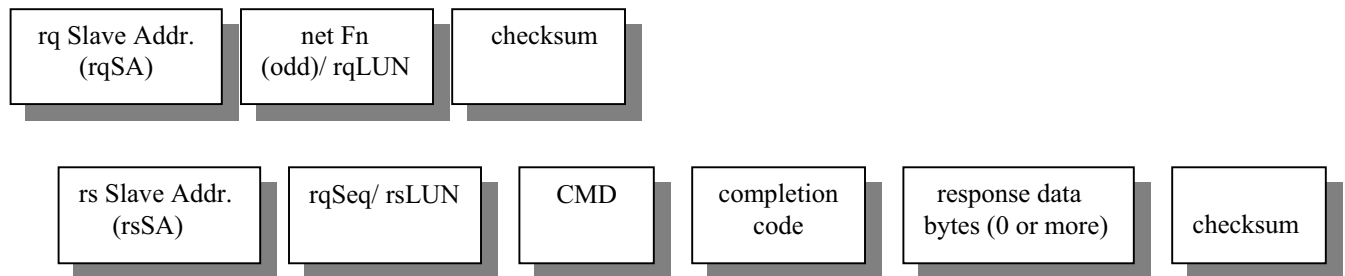
## 1.0 Overview

This specification describes the implementation of an IPMI/IPMB Power Supply satellite controller. This controller will provide an interface to sensors to monitor the power supply and will physically attach to the IPMB (I<sup>2</sup>C) bus. The satellite controller will process IPMI commands that are communicated to the controller via the IPMB bus protocol. This is a request-response protocol that uses the I<sup>2</sup>C bus as the physical interface. The general structure of the IPMB protocol is shown below

### Request:



### Response:



**Figure 1-1. IPMB Request/Response Protocol**

This format is the same for all IPMB commands except Broadcast commands. Broadcast commands (i.e., Broadcast Get Device ID) are preceded by a byte of all 0's. The restrictions on I<sup>2</sup>C services required by this protocol are given as follows:

1. Only Master Write transactions are used (R/W bit on slave address is always 0).
2. 10-bit addressing is not used.
3. Only one slave address is used in a single transaction.
4. No repeated starts are used. Repeated starts are defined in the I<sup>2</sup>C specification as a means of changing bus direction during a transfer. Refer to the references on I<sup>2</sup>C for more information. Since the protocol only uses Master Write transactions, repeated starts are not required.

Note, that while repeated starts are not used in Intelligent Platform Management Bus transactions they can be used for non-protocol transactions on the same bus. I.e., you can still use repeated starts to access devices that don't use the IPMB protocol.

5. The I<sup>2</sup>C acknowledgment mechanism ('ACK' bit) only indicates acceptance of the byte by a slave and does not convey any additional information regarding the integrity or correctness of the received data.

The notation and definitions of the protocol elements is given below in Table 1-2.

<b>Designation</b>	<b>Meaning</b>
chk, checksum	2's complement checksum of preceding bytes in the connection header or between the previous checksum. 8-bit checksum algorithm: Initialize checksum to 0. For each byte, checksum = (checksum + byte) modulo 256. Then checksum = - checksum. When the checksum and the bytes are added together, modulo 256, the result should be 0.
cmd, command	Command Byte(s) - one or more command bytes -as required by the network identify the type of request
data	As required by the particular request or response
LUN	The lower 2-bits of the netFn byte identify the logical unit number, which provides further sub-addressing within the target node.
netFn	6-bit value that selects the set of functions (command set) within a target node to be accessed. The value is even if a request, odd if a response. For example a value of 02h means a bridging request, 03h a bridging response. Refer to Section 3, Network Functions and Commands, below for more information.
rq	Abbreviation for 'Requester'.
rq XNA	Requester's External Node Address. 3 bytes.
rqLUN	Requester's LUN.
rqSA	Requester's Slave Address. 1 byte. LS bit always 0.
rs	Abbreviation for 'Responder'.
rsLUN	Responder's LUN
rsSA	Responder's Slave Address. 1 byte. LS bit always 0.
Seq	Sequence field. This field is used to verify that a response is for a particular instance of a request.
XX.YY	Denotes the combination of netFn and CMD that is ultimately received at the node once any bridging headers and LUN have been stripped off.

**Table 1-2 Protocol Definitions**

The Network Function (netFn) codes used in IPMI are defined in Table 1-3. The Network Functions that will be recognized by this controller are the application (app) functions, the Sensor/Event (S/E) functions, Firmware functions, and OEM defined functions.

<b>netFn</b>	<b>Name</b>	<b>Meaning</b>	<b>Description</b>
00,01	Chassis	Chassis Device Requests and Responses	00 – Request 01 - Response
02, 03	Bridges	Bridge requests and Responses	02 – Request 03 - Response
04, 05	Sensor/ Event	Sensor and Event Requests and Responses	04 – Request 05 - Response
06,07	App	Application Requests and Responses	06 – Request 07 - Response
08,09	Firmware	Firmware Transfer Requests and Responses	08 – Request 09 - Response
0A, 0B	Storage	Non-volatile Storage Requests and Responses	0A – Request 0B - Response
0Ah - 2Fh	Reserved	-	Reserved
30h – 3Fh	OEM	-	Vendor Specific

**Table 1-3 Network Function (netFn) Definitions**

## 2. Implementation

### 2.1 Communications Processing

Referring to Figure 1-1, the first 5 bytes of the IPMB message contain only addressing, error checking, and network function information that are common to all IPMB messages except broadcast messages. Broadcast messages are the same format except they are preceded by a byte of all 0's. As a result, the addressing, error checking, and network function portion of all messages can be processed without knowing the specific command. The first byte in the request is the I<sup>2</sup>C destination address of the responder that is placed in the fourth byte of the response. The fourth byte of the request is the I<sup>2</sup>C address of the requester and is placed in the first byte of the response. The second byte of the request contains the network function (netFn) information and the responder's logical unit number (LUN). The fourth byte of the request has the requester's sequence number and requester's LUN. The responder's LUN is extracted from the second byte of the request and combined with the original requester's sequence number in the fourth byte of the response. The netFn for a request is always even, and the netFn for a response is always odd. Referring to table 1-3, it can be observed that without interpreting the network function, the network function placed in the response can be obtained by incrementing the request netFn. This response netFn (odd) is combined with the requester's LUN in the second byte of the response. Lastly, byte 3 of the request contains a checksum that is the 2's complement of the sum of the first two bytes. A new checksum is generated for the third byte of the response that is the 2's complement of the sum of the first two bytes of the response. This controller will only acknowledge messages with matching I<sup>2</sup>C address or broadcast messages.

### 2.2 Command Processing

The fifth byte of the IPMB request is the command being issued. In order to properly interpret this command, the preceding netFn must also be considered. The netFn defines the command set used. As a result, command processing is sorted by the netfunction field, processing "application" network function commands, processing "sensor/event" commands, processing "Firmware" Commands, and processing "OEM" commands. The IPMI/IPMB satellite controller described here has the following commands implemented:

Application commands:

- Get Device ID
- Broadcast Get Device ID
- Cold Reset
- Warm Reset
- Get Self Test Results

#### Sensor/Event Commands:

- Set Sensor Hysteresis
- Get Sensor Hysteresis
- Set Sensor Thresholds
- Get Sensor Thresholds
- Get Sensor Event Status
- Get Sensor Reading
- Set Event Enables
- Get Event Enables
- Set Event Receiver
- Get Event Receiver
- Get Device SDR Information
- Get Device SDR
- Reserve Device Repository

#### Firmware Commands:

- Start Firmware Update
- Erase Flash
- Program Flash
- Exit Firmware Update

#### OEM Commands:

- Get Device Status
- Set Fault LED State
- Get Fault LED State
- Set Inhibit State
- Get Inhibit State

OEM Vendor Only Commands: (used to program serial numbers, device ID, etc into flash)

- Program Device ID
- Lock Programmed Data

The implementation detail of these commands is described in the following sections. Each of the command requests supported will be detailed. An example of the request and response format of the commands and a table with the details of the information inside the request and response messages is given. The format for the example figures is the same as the format used in Section 5.1 (Figure 5.1) in the Intelligent Platform Management Bus Communications Protocol Specification, V1.0. The format for the tables is the same as the format for each of the detailed commands in the Intelligent Platform Management Interface Specification, V1.5.

## 2.3 I/O Detail

MNEMONIC	PIN NO.	TYPE	NAME AND FUNCTION
1VL		I	Voltage Input 1
3VL		I	Voltage Input 3
4VL		I	Voltage Input 4
5VL		I	Voltage Input 5
-IS1		I	Current Input 1 (current inputs 1 and 2 are summed together)
-IS2		I	Current Input 2 “ “
-IS3		I	Current Input 3
-IS4		I	Current Input 4
IS5		I	Current Input 5
IS		I	Common return for differential current inputs
A TH		I	Analog thermal sensor
VCCA		I	Board +5V Power
GND		I/O	Ground
AUX1			Reserved for future use
AUX2			Reserved for future use
EN		I	Enable Input – Active Low to Enable
INH		I	Inhibit Input – Active Low to Inhibit. Takes precedence over EN
FAL		O	Fault Output – TTL Open Collector Out, Active Low
ADR2		I	Slot Address Input 2 – Normally low for no address offset
ADR1		I	Slot Address Input 1 – Normally low for no address offset
ADR0		I	Slot Address Input 0 – Normally low for no address offset
VCCB		I	Bus Vcc
SDA		I/O	IPMB/I2C SDA
SCL		I/O	IPMB/I2C SCL

## 2.4 LED Functionality

### 2.4.1 Input LED – Green

ON – Input power present, Outputs Enabled  
 Blinking ON – Input power present, Output Inhibited  
 OFF – Input power not present

### 2.4.2 Fault LED – Amber

ON – One of the Outputs has exceeded an alarm threshold.  
 OFF – Normal Operation  
 OVERRIDE – The state of this LED can also be set by an OEM IPMI command. See the description for the “Set Fault LED State” command in Section 7, OEM Commands, for details.

### 3. Application Commands

#### 3.1 Get Device ID Command

Figure 3.1 shows the structure and example data for a Get Device ID request and response. The detail of the request and response data after the command byte is given in Table 3-1. A detailed breakdown of the Device ID Information is given in Appendix A. This information is programmed into non-volatile memory using the “Program Device ID Information” command detailed in section 6.3.1.

#### 3.2 Broadcast Get Device ID Command

The Broadcast Get device ID command is the same as the Get Device ID command except is preceded by a byte of all zeros. The response to the Broadcast Get device ID command is the same as for the regular Get Device ID command. The format with an example is given in Figure 3.2

**Figure 3.1: Get Device ID Command**  
Bits

Request:

	7	6	5	4	3	2	1	0	
Byte 1	<b>rsSA = 52h</b>								52h
Byte2	<b>netFn = 06h</b> (app request)				<b>rsLUN = 00</b>				18h
Byte3	<b>checksum = 96h</b>								96h
Byte4	<b>rqSA = 20h</b>								20h
Byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h
Byte6	<b>cmd = 01h</b> (get device id)								01h
Byte7	<b>checksum = DBh</b>								DBh

Response:

	7	6	5	4	3	2	1	0	
byte 1	<b>rqSA = 20h</b>								20h
byte 2	<b>netFn = 07h</b> (app response)				<b>rqLUN = 00</b>				1Ch
byte 3	<b>checksum = C4h</b>								C4h
byte 4	<b>rsSA = 52h</b>								52h
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h
byte 6	<b>cmd = 01h</b> (get device id)								01h
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h
byte 8	<b>data byte 2 = 00h</b> (device id)								00h
byte 9	<b>data byte 3 = 81h</b> (device rev)								81h
byte 10	<b>data byte 4 = 01h</b> (major f/w rev)								01h
byte 11	<b>data byte 5 = 01h</b> (minor f/w rev)								01h
byte 12	<b>data byte 6 = 51h</b> (IPMI version. V1.5 = 51h)								51h
byte 13	<b>data byte 7 = 21h</b> (device support. EG&sensors)								21h
byte 14	<b>data byte 8 = xx</b> (Manufacturer ID, LS byte)								71h
byte 15	<b>data byte 9 = xx</b> (Manufacturer ID, middle byte)								33h
byte 16	<b>data byte 10 = xx</b> (Manufacturer ID, MS byte)								00h
byte 17	<b>data byte 11 = xxh</b> (product ID, LS byte)								16h
byte 18	<b>data byte 12 = xxh</b> (Product ID, MS byte)								13h
byte 19	<b>checksum = XXh</b>								E7h

**Table 3-1. Get Device ID Detail.**

Request Data  
Response Data

-	-
1	Completion Code
2	Device ID. 00h = unspecified.
3	Device Revision [7] 1 = device provides Device SDRs 0 = device does not provide Device SDRs [6:4] reserved. Return as 0. [3:0] Device Revision, binary encoded.
4	Firmware Revision 1 [7] Device available: 0=normal operation, 1= device firmware, SDR Repository update or self-initialization in progress. [Firmware / SDR Repository updates can be differentiated by issuing a Get SDR command and checking the completion code.] [6:0] Major Firmware Revision, binary encoded.
5	Firmware Revision 2: Minor Firmware Revision. BCD encoded.
6	IPMI Version. Holds IPMI Command Specification Version. BCD encoded. 00h = reserved. Bits 7:4 hold the Least Significant digit of the revision, while bits 3:0 hold the Most Significant bits. E.g. a value of <b>51h</b> indicates revision 1.5.
7	Additional Device Support (formerly called IPM Device Support). Lists the IPMI 'logical device' commands and functions that the controller supports that are in addition to the mandatory IPM and Application commands. [7] Chassis Device (device functions as chassis device per ICMB spec.) [6] Bridge [5] IPMB Event Generator (device generates event messages [platform event request messages] onto the IPMB) [4] IPMB Event Receiver (device accepts event messages [platform event request messages] from the IPMB) [3] FRU Inventory Device [2] SEL Device [1] SDR Repository Device [0] Sensor Device
8:10	Manufacturer ID, LS Byte first. The manufacturer ID is a 20-bit value that is derived from the IANA 'Private Enterprise' ID (see below). Most significant four bits = reserved (0000b). 000000h = unspecified. 0FFFFFFh = reserved. This value is binary encoded. E.g. the ID for the IPMI forum is 7154 decimal, which is 1BF2h, which would be stored in this record as F2h, 1Bh, 00h for bytes 8 through 10, respectively.
11:12	Product ID, LS Byte first. This field can be used to provide a number that identifies a particular system, module, add-in card, or board set. The number is specified according to the manufacturer given by Manufacturer ID (see below). 0000h = unspecified. FFFFh = reserved.
13:16	Auxiliary Firmware Revision Information. This field is optional. If present, it holds additional information about the firmware revision, such as boot block or internal data structure version numbers. The meanings of the numbers are specific to the vendor identified by Manufacturer ID (see below). When the vendor-specific definition is not known, generic utilities should display each byte as 2-digit hexadecimal numbers, with byte 13 displayed first as the most-significant byte.

**Figure 3.2 Broadcast Get Device ID Command:**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
Byte 1	<b>Broadcast = 00h</b>								00h		
Byte 2	<b>rsSA = 52h</b>								52h		
Byte 3	<b>netFn = 06h</b> (app request)				<b>rsLUN = 00</b>				18h		
Byte 4	<b>checksum = 96h</b>								96h		
Byte 5	<b>rqSA = 20h</b>								20h		
Byte 6	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
Byte 7	<b>cmd = 01h</b> (get device id)								01h		
Byte 8	<b>checksum = DBh</b>								DBh		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 07h</b> (app response)				<b>rqLUN = 00</b>				1Ch		
byte 3	<b>checksum = C4h</b>								C4h		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 01h</b> (get device id)								01h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>data byte 2 = 00h</b> (device id)								00h		
byte 9	<b>data byte 3 = 81h</b> (device rev)								81h		
byte 10	<b>data byte 4 = 01h</b> (major f/w rev)								01h		
byte 11	<b>data byte 5 = 01h</b> (minor f/w rev)								01h		
byte 12	<b>data byte 6 = 51h</b> (IPMI version. V1.5 = 51h)								51h		
byte 13	<b>data byte 7 = 21h</b> (device support. EG&sensors)								21h		
byte 14	<b>data byte 8 = xx</b> (Manufacturer ID, LS byte)								71h		
byte 15	<b>data byte 9 = xx</b> (Manufacturer ID, middle byte)								33h		
byte 16	<b>data byte 10 = xx</b> (Manufacturer ID, MS byte)								00h		
byte 17	<b>data byte 11 = xxh</b> (product ID, LS byte)								16h		
byte 18	<b>data byte 12 = xxh</b> (Product ID, MS byte)								13h		
byte 19	<b>checksum = xxh</b>								E7h		

### 3.3 Cold Reset Command

Upon receiving and validating this command, the processor will be reset, giving the same result as a power on reset. A detailed example of format this command is given in figure 3-3, with the command detail in table 3-3.

**Figure 3-3 Cold Reset**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 06h</b> (app request)				<b>rsLUN = 00</b>				18h		
byte3	<b>checksum = 96h</b>								96h		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 02h</b> (Cold Reset)								02h		
byte7	<b>checksum = DAh</b>								DAh		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 07h</b> (app response)				<b>rqLUN = 00</b>				1Ch		
byte 3	<b>checksum = C4h</b>								C4h		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 02h</b> (Cold Reset)								02h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>checksum = A8h</b>								A8h		

**Table 3-3 Cold Reset**

Request Data	-	-
Response Data	1	Completion Code

Note from the IPMI specification: The *Cold Reset* command is provided for platform development, test, and platform-specific initialization and recovery actions. The system actions of the *Cold Reset* command are platform specific. Issuing a *Cold Reset* command could have adverse effects on system operation, particularly if issued during run-time. Therefore, the *Cold Reset* command should not be used unless all the side-effects for the given platform are known.

It is recognized that there are conditions where a given controller may not be able to return a response to a *Cold Reset* Request message. Therefore, though recommended, the implementation is not required to return a response to the *Cold Reset* command. Applications should not rely on receiving a response as verification of the completion of a *Cold Reset* command.

This application attempts to return a response to the warm and cold reset commands, however, since the processor is being reset, a complete response may not always be returned.

### 3.4 Warm Reset Command

This command causes the processor to start at the beginning of the program which causes the initialization and start up function to be executed. A detailed example of this command is given in figure 3.4 and table 3.4.

**Figure 3.4 - Warm Reset:**

**Request:**

		Bits									
		7	6	5	4	3	2	1	0		
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 06h</b> (app request)				<b>rsLUN = 00</b>				18h		
byte3	<b>checksum = 96h</b>								96h		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 03h</b> (Warm Reset)								03h		
byte7	<b>checksum = D9h</b>								D9h		

**Response:**

		Bits									
		7	6	5	4	3	2	1	0		
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 07h</b> (app response)				<b>rqLUN = 00</b>				1Ch		
byte 3	<b>checksum = C4h</b>								C4h		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 03h</b> (Warm Reset)								03h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>checksum = A7h</b>								A7h		

**Table 3-3 Warm Reset**

Request Data	-	-
Response Data	1	Completion Code

**3.4 Get Self Test Results command**

This command instructs the controller to give the status of a self test run by the controller. The detail of the request and response data after the command byte is given in Table 2-2 . Most of the failure modes listed pertain to the BMC and not the satellite controller. As a result, the self-test data reported in this response, should be considered device specific. For this application the self-test function will check the operation of the power supply and not the device itself. This self-test verifies that none of the sensors are exceeding the failure thresholds. The IPMB format of this command with an example is given in Figure 2.3.

Request Data	-	-
Response Data	1	Completion Code
	2	<p>55h = No error. All Self Tests Passed.                      56h = Self Test function not implemented in this controller.                      57h = Corrupted or inaccessible data or devices                      58h = Fatal hardware error (system should consider BMC inoperative). This will indicate that the controller hardware (including associated devices such as sensor hardware or RAM) may need to be repaired or replaced.                      FFh = reserved.                      all other: Device-specific 'internal' failure. Refer to the particular device's specification for definition.</p> <p><b>For the Satellite controller application (Device specific):</b>  <b>55h = Power supply outputs are OK.</b>  <b>59h = A power supply sensor has exceeded a failure threshold.</b></p>
	3	<p>For byte 2 = 55h, 56h, FFh: 00h                      For byte 2 = 58h, all other: Device-specific                      For byte 2 = 57h: self-test error bit field. Note: returning 57h does not imply that all tests were run, just that a given test has failed. I.e. 1b means 'failed', 0b means 'unknown'.</p> <p>[7] 1b = Cannot access SEL device                      [6] 1b = Cannot access SDR Repository                      [5] 1b = Cannot access BMC FRU device                      [4] 1b = IPMB signal lines do not respond                      [3] 1b = SDR Repository empty                      [2] 1b = Internal Use Area of BMC FRU corrupted                      [1] 1b = controller update 'boot block' firmware corrupted                      [0] 1b = controller operational firmware corrupted</p> <p><b>For this Satellite controller application (Device specific):</b>  <b>Byte 3 = 00h</b></p>

**Table 3-4. Get Self Test Results Command**

**Figure 3-4. Get Self Test Results Command:**

**Request:**

		Bits							
		7 6 5 4 3 2 1 0							
byte1	<b>rsSA = 52h</b>								52h
byte2	<b>netFn = 06h</b> (app request)				<b>rsLUN = 00</b>				18h
byte3	<b>checksum = 96h</b>								96h
byte4	<b>rqSA = 20h</b>								20h
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h
byte6	<b>cmd = 04h</b> (get self test results)								04h
byte7	<b>checksum = D8h</b>								D8h

**Response:**

		Bits							
		7 6 5 4 3 2 1 0							
byte 1	<b>rqSA = 20h</b>								20h
byte 2	<b>netFn = 07h</b> (app response)				<b>rqLUN = 00</b>				1Ch
byte 3	<b>checksum = A0h</b>								A0h
byte 4	<b>rsSA = 52h</b>								52h
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h
byte 6	<b>cmd = 04h</b> (get self test results)								04h
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h
byte 8	<b>data byte 2 = 55h</b> (self test results – passed)								55h
byte 9	<b>data byte 3 = 00h</b> (error codes - none)								00h
byte 10	<b>checksum = 8Ah</b>								51h

Last update:12-15-03

## 4. Sensor Event Commands

The Sensor and Event commands accomplish a number of tasks including:

- Providing access to Sensor Data Records (SDRs) - The SDR is a series of records that detail the specific type sensor, how the sensor is implemented, and conversion factors to interpret “raw” sensor data reads.
- Provides access to the sensors and ability to obtain the latest sensor reading.
- Provides the capability to generate an “Event Message” if a sensor exceeds one of several thresholds.
- Provides the ability to program the thresholds and hysteresis values used in generating the Event Messages.
- Provides the ability to enable or disable the event generation capability of a sensor or sensors, and/or, one of several of the programmable thresholds.
- Provides the ability to enable or disable the event generation capability of one or several of the programmable thresholds within a sensor.

### 4.1 Get Device SDR Info Command

The Get Device SDR Info command provides the number of sensors addressed under the specified LUN. It also provides information on whether the sensors are static or dynamic. If the sensor is dynamic, the command also provides (as part of the response) a sensor population change indicator.

Static Sensor Devices are defined as sensors that have their Sensor Data Records added to the SDR Repository when the device is configured into the system. This is normally done either as part of the manufacturing of the system, or via a separate utility when they are added to or deleted from the system configuration.

Dynamic Sensor Devices rely on being *discovered* by responding to a *Broadcast Get Device ID* formatted to their slave address. (The IPMB format of this message is identical to that for a *Get Device ID* Request message that has the entire message prefixed with the I<sup>2</sup>C broadcast slave address. [00h]) Once discovered, dynamic sensor devices can be queried for their sensor population via the *Get Device SDR Info* command.

For this application, this device’s sensor population is static because the number of sensors for this device is fixed, and a query shall return records for all sensors. A sensor population change indicator will not be provided in the response. However, this sensor will respond to the same discovery commands as a dynamic sensor. The command detail is given in Table 4.1, with the IPMB format of this command with an example is given in Figure 4.1.

Request Data  
Response Data

-	-
1	Completion Code
2	Number of sensors in device for LUN this command was addressed to.
3	<p>flags:</p> <p style="text-align: right;">Dynamic population</p> <p>[7] - 0b = static sensor population. The number of sensors handled by this device is fixed, and a query shall return records for all sensors.</p> <p style="padding-left: 2em;">1b = dynamic sensor population. This device may have its sensor population vary during 'run time' (defined as any time other than when an install operation is in progress).</p> <p style="text-align: right;">Reserved</p> <p>[6:4] - reserved</p> <p style="text-align: right;">Device LUNs</p> <p>[3] - 1b = LUN 3 has sensors  [2] - 1b = LUN 2 has sensors  [1] - 1b = LUN 1 has sensors  [0] - 1b = LUN 0 has sensors</p>
4:7	<p>Sensor Population Change Indicator. LS byte first.</p> <p>Four byte timestamp, or counter. Updated or incremented each time the sensor population changes. This field is not provided if the flags indicate a static sensor population.</p>

**Table 4-1 Get Device SDR Information**

**Figure 4.1 Get Device SDR Information Command**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 04h</b> (S/E Request)				<b>rsLUN = 00</b>				10h		
byte3	<b>checksum = 9Eh</b>								9Eh		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 20h</b> (Get SDR Information)								20h		
byte7	<b>Checksum = BCh</b>								BCh		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>				14h		
byte 3	<b>checksum = CCh</b>								CCh		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 20h</b> (Get SDR Information)								20h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>data byte 2 = 09h</b> (# of sensors for this LUN)								09h		
byte 9	<b>data byte 3 = 01h</b> (Flags)								01h		
byte 10	<b>checksum = 80h</b>								80h		

#### 4.2 Get Device SDR Command

The 'Get Device SDR' command allows the population of sensors for the given LUN in the device to be listed and the associated SDR information for those sensors returned.

Sensor Devices that support the 'Get Device SDR' command return SDR Records that match the SDR Repository formats. See Tables 37-1 and 37-7 in Appendix B, *Sensor Data Record Formats*.

A tabular version of the Sensor Data Records is shown in table 4-3.

The 'Get Device SDR' command includes a *Reservation ID* that is used to notify the Requester that a record may have changed during the process of a multi-part read. For this application with a static sensor population, the *Reservation ID* will not change. The command detail is given in Table 4.2.1, with the IPMB format of this command with an example is given in Figure 4.2.

**Table 4.2.1 Get Device SDR Information**

Request Data	1	Reservation ID. LS Byte. Only required for partial reads with a non-zero 'Offset into record' field. Use 0000h for reservation ID otherwise.
	2	Reservation ID. MS Byte.
	3	Record ID of record to Get, LS Byte. 0000h returns the first record.
	4	Record ID of record to Get, MS Byte
	5	Offset into record
	6	Bytes to read. FFh means read entire record.
Response Data	1	Completion Code. Generic, plus following command specific: 80h = record changed. This status is returned if any of the record contents have been altered since the last time the Requester issued the request with 00h for the 'Offset into SDR' field.
	2	Record ID for next record, LS Byte
	3	Record ID for next record, MS Byte
	4:3+n	Requested bytes from record

This application will use two types of SDRs, Type 01h – Full Sensor Record, and Type 12h – IPMB Management Controller Device Locator.

The information in the IPMB Management Controller Device Locator (Type 12h ) is used for identifying management controllers on the IPMB and other internal channels, and for providing Entity and initialization information for all management controllers. Table 4-2.2 shows the fields and values given for the Type 12h SDR for this application, With appendix B giving a detailed breakdown of each or the fields in the Type 12h SDR. The values used in this application are given in Table 4.2.2.

The Full Sensor Record (Type 01h) can be used to describe any type of sensor. Each sensor has it's own Type 01h record that describes all of the characteristics and capabilities of the sensor. Appendix B gives a detailed breakdown of each or the fields in the Type 01h SDR. The values used in this application are given in Table 4.2.3.

**Figure 4-2 Get Device SDR Command Example**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rsSA = 52h</b>								52h		
byte 2	<b>netFn = 04h</b> (S/E request)				<b>rsLUN = 00</b>				10h		
byte 3	<b>checksum = 9Eh</b>								9Eh		
byte 4	<b>rqSA = 20h</b>								20h		
byte 5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte 6	<b>cmd = 21h</b> (Get Device SDR)								21h		
byte 7	<b>byte 1 = 00h</b> (Reservation ID, LS Byte)								01h		
byte 8	<b>byte 2 = 00h</b> (Reservation ID, MS Byte)								00h		
byte 9	<b>byte 3 = 00h</b> (Record ID of Record, LS Byte)								00h		
byte 10	<b>byte 4 = 00h</b> (Record ID of Record, MS Byte)								00h		
byte 11	<b>byte 5 = 00h</b> (Offset into Record)								00h		
byte 12	<b>byte 2 = 10h</b> (Bytes to read)								10h		
byte 13	<b>checksum = AAh</b>								AAh		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>				14h		
byte 3	<b>checksum = CCh</b>								CCh		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 21h</b> (Get Device SDR)								21h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>data byte 2 = 01h</b> (ID of next Record, LS Byte)								01h		
byte 9	<b>data byte 3 = 00h</b> (ID of next Record, MS Byte)								00h		
byte 10	<b>data byte 4:N =</b> (Requested bytes from Record)								xxh		
byte 11	<b>checksum = xx h</b>								xxh		

**Table 4.2.2: Type 12 SDR (Management Controller Device Locator):**

Byte	Name	Value	Notes
1	Record ID – L	00h	Make it the first record to be read
2	Record ID – H	00h	
3	SDR Version	51h	IPMI V1.5
4	Record Type	12h	Type 12h SDR
5	Record Length	0Dh	(13)
6	Device Slave Address	52h+	Note 1
7	Channel Number	00h	see table 18-10, p193
8	Power State Notification - Global Initialization	24h	
9	Device Capabilities	2Bh	Event Generator, Sensors, FRU, SDR
10	reserved	00h	
11	reserved	00h	
12	reserved	00h	
13	Entity ID	0Ah	
14	Entity Instance	off/2	Note 2
15	OEM	FFh	
16	Device ID String - Type/Length	02h	
17	Device ID String-L	30h	Only 2 bytes are used for this application
18	Device ID String-H	31h	(ASCII 0,1)

**SDR Notes:**

**Note 1-** The device Slave address is calculated based on the base slave address (52h) plus any offset indicated when reading the connector slot location state (inputs ADR0, ADR1, ADR2). Depending on the state of these pins, the address offset will be 00h, 02h, 04h, 06h, 08h, 0Ah, 0Ch, or 0Eh)

**Note 2** – The instance number will also be calculated by reading the connector slot location state (inputs ADR0, ADR1, ADR2). The offset will be divided by two to calculate the instance number. This will result in instance numbers 00h, 01h, 02h, 03h, 04h, 05h, and 06h, depending on the slot location.

**Note 3** – Sensors 00, 01, 02, and 03 correspond to voltage inputs 1VL (5.0V nom), 2VL(3.3v nom), 3VL (12V nom), and 4VL(-12V nom) respectively. Sensors 04, 05, 06, 07 correspond to IS1 (30A nom), IS2 (40A nom), IS3 (5A nom), and IS4 (-1A nom) respectively. Sensor 8 is the analog thermal sensor.

**Note 4** - The conversion formula for converting “raw” sensor data into the final reading is given in Appendix F.

**Table 4.2.3, Type 01 SDR:**

Full Sensor Record		Sensor Number									Notes
Byte	Name	00	01	02	03	04	05	06	07	08	
1	Record ID – L	01h	02h	03h	04h	05h	06h	07h	08h	09h	
2	Record ID – H	00h	00h	00h	00h	00h	00h	00h	00h	00h	
3	SDR Version	51h	51h	51h	51h	51h	51h	51h	51h	51h	
4	Record Type	01h	01h	01h	01h	01h	01h	01h	01h	01h	
5	Record Length	3Bh	3Bh	3Bh	3Bh	3Bh	3Bh	3Bh	3Bh	3Bh	
6	Sensor Owner ID	52h+	52h+	52h+	52h+	52h+	52h+	52h+	52h+	52h+	1
7	Sensor Owner LUN	00h	00h	00h	00h	00h	00h	00h	00h	00h	
8	Sensor Number	00h	01h	02h	03h	04h	05h	06h	07h	08h	3
9	Entity ID	0Ah	0Ah	0Ah	0Ah	0Ah	0Ah	0Ah	0Ah	0Ah	
10	Entity Instance	off/2	off/2	off/2	off/2	off/2	off/2	off/2	off/2	off/2	2
11	Sensor Initialization	03h	03h	03h	03h	03h	03h	03h	03h	03h	
12	Sensor Capabilities	E8h	E8h	E8h	E8h	E8h	E8h	E8h	E8h	E8h	
13	Sensor Type	02h	02h	02h	02h	03h	03h	03h	03h	01h	
14	Event / Reading Type Code	01h	01h	01h	01h	01h	01h	01h	01h	01h	
15:16	Assertion Event Mask / LowerThreshold Reading Mask	7Ah	7Ah	7Ah	7Ah	0Ah	0Ah	0Ah	0Ah	0Ah	
		95h	95h	95h	95h	80h	80h	80h	80h	80h	
17:18	Deassertion Event Mask / Upper Threshold Reading Mask	7Ah	7Ah	7Ah	7Ah	7Ah	7Ah	7Ah	7Ah	7Ah	
		95h	95h	95h	95h	80h	80h	80h	80h	80h	
19:20	Discrete Reading Mask / SettableThreshold Mask, ReadableThreshold Mask	3Fh	3Fh	3Fh	3Fh	38h	38h	38h	38h	38h	
		3Fh	3Fh	3Fh	3Fh	38h	38h	38h	38h	38h	
21	Sensor Units 1	00h	00h	00h	00h	00h	00h	00h	00h	00h	
22	Sensor Units 2 - Base	04h	04h	04h	04h	05h	05h	05h	05h	01h	
23	Sensor Units 3 - Modifier	00h	00h	00h	00h	00h	00h	00h	00h	00h	
24	Linearization	00h	00h	00h	00h	00h	00h	00h	00h	00h	
25	M	0Bh	B0h	40h	C0h	95h	B4h	0Ah	CCh	3Ah	
26	M, Tolerance	42h	02h	02h	C2h	04h	04h	44h	C4h	06h	
27	B	00h	00h	00h	00h	00h	00h	00h	00h	00h	
28	B, Accuracy	16h	16h	16h	16h	16h	16h	16h	16h	16h	
29	Accuracy, Accuracy exp	20h	20h	20h	20h	20h	20h	20h	20h	20h	
30	R exp, B exp	C0h	C0h	D0h	D0h	D0h	D0h	C0h	C0h	E0h	
31	Analog characteristic flags	01h	01h	01h	01h	01h	01h	01h	01h	01h	
32	Nominal Reading	C6h	BFh	C2h	BFh	C0h	C0h	C0h	C0h	86h	
33	Normal Maximum	C8h	C8h	C4h	BFh	CBh	DDh	C3h	C1h	FFh	
34	Normal Minimum	C3h	Bh	C1h	BEh	0Bh	04h	02h	00h	00h	
35	Sensor Maximum Reading	FFh	FFh	FFh	FFh	FFh	FFh	FFh	FFh	FFh	
36	Sensor Minimum Reading	00h	00h	00h	00h	00h	00h	00h	00h	00h	
37	Upper non-recoverable Threshold	D8h	D8h	D8h	D8h	D8h	D8h	D8h	D8h	FFh	
38	Upper critical Threshold	CEh	CEh	CEh	CEh	CEh	CEh	CEh	CEh	FFh	
39	Upper non-critical Threshold	C5h	C5h	C5h	C5h	C5h	C5h	C5h	C5h	FFh	
40	Lower non-recoverable Threshold	9Fh	9Fh	9Fh	9Fh	00h	00h	00h	00h	00h	
41	Lower critical Threshold	A9h	A9h	A9h	A9h	00h	00h	00h	00h	00h	
42	Lower non-critical Threshold	B2h	B2h	B2h	B2h	00h	00h	00h	00h	00h	
43	positive-going Threshold Hysteresis value	04h	04h	04h	04h	04h	04h	04h	04h	04h	
44	Negative-going Threshold Hysteresis value	04h	04h	04h	04h	04h	04h	04h	04h	04h	
45	reserved	00h	00h	00h	00h	00h	00h	00h	00h	00h	
46	reserved	00h	00h	00h	00h	00h	00h	00h	00h	00h	
47	OEM	FFh	FFh	FFh	FFh	FFh	FFh	FFh	FFh	FFh	
48	ID String Type/Length Code	CBh	CDh	CCh	CCh	CEh	D0h	CFh	CFh	CEh	

**Table 4.2.3, Type 01 SDR:**

Full Sensor Record (cont)		Sensor Number									Notes
Byte	Name	00	01	02	03	04	05	06	07	08	
49	ID String - byte 1	53h	53h	53h	53h	53h	53h	53h	53h	53h	5
50	ID String - byte 2	30h	31h	32h	33h	34h	35h	36h	37h	38h	5
51	ID String - byte 3	20h	20h	20h	20h	20h	20h	20h	20h	20h	5
52	ID String - byte 4	2Bh	2Bh	2Bh	2Dh	2Bh	2Bh	2Bh	2Dh	54h	5
53	ID String - byte 5	35h	33h	31h	31h	35h	33h	31h	31h	65h	5
54	ID String - byte 6	56h	2Eh	32h	32h	56h	2Eh	32h	32h	60h	5
55	ID String - byte 7	20h	33h	56h	56h	20h	33h	56h	56h	70h	5
56	ID String - byte 8	28h	56h	20h	20h	43h	56h	20h	20h	65h	5
57	ID String - byte 9	56h	20h	28h	28h	75h	20h	43h	43h	72h	5
58	ID String - byte 10	31h	28h	56h	56h	72h	75h	72h	72h	74h	5
59	ID String - byte 11	29h	56h	33h	34h	72h	75h	72h	72h	75h	5
60	ID String - byte 12	20h	32h	29h	65h	72h	43h	75h	75h	61h	5
61	ID String - byte 13	20h	29h	20h	20h	6Eh	72h	72h	62h	75h	5
62	ID String - byte 14	20h	20h	20h	20h	74h	65h	6Eh	6Eh	65h	5
63	ID String - byte 15	20h	20h	20h	20h	20h	6Eh	74h	74h	20h	5
64	ID String - byte 16	20h	20h	20h	20h	20h	74h	20h	20h	20h	5

SDR Notes (cont):

**Note 5** – ID String bytes, 8 bit ASCII – The last 16 bytes of the Type 1 SDR contain an ASCII string with a description of the sensor. The ASCII hex values in the table translate to the following text strings:

- Sensor 00 – “S0 +5V (V1)”
- Sensor 01 – “S1 +3.3V (V2)”
- Sensor 02 – “S2 +12V (V3)”
- Sensor 03 – “S3 -12V (V4)”
- Sensor 04 – “S4 +5V Current”
- Sensor 05 – “S5 +3.3V Current”
- Sensor 06 – “S6 +12V Current”
- Sensor 07 – “S7 -12V Current”
- Sensor 08 – “S8 Temperature”

### 4.3 Get Device SDR Repository Command

This command is used to obtain a *Reservation ID*. The Reservation ID is part of a mechanism that is used to notify the Requester that a record may have changed during the process of a multi-part read. Since this device has a static sensor population, the Reservation ID is fixed and will not change. The command detail is given in Table 4.3, with the IPMB format of this command with an example is given in Figure 4.3.

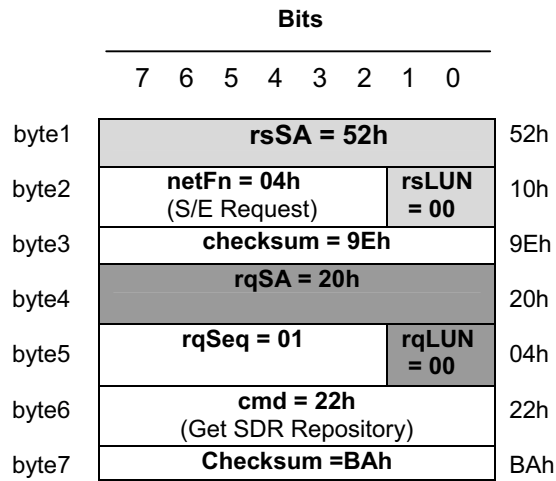
**Table 4-3 Get Device SDR Repository Command**

Request Data  
Response Data

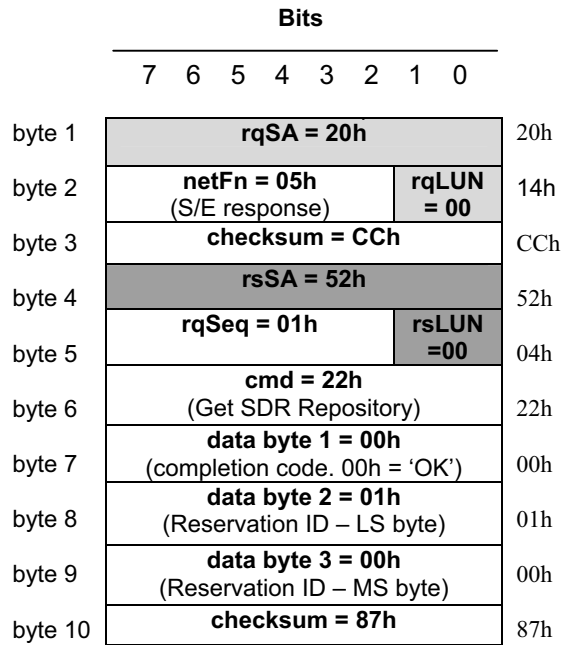
-	-
1	Completion Code
2	Reservation ID, LS Byte 0000h reserved.
3	Reservation ID, MS Byte

**Figure 4.3 Get SDR Repository Command**

Request:



Response:



#### 4.4 Set Event Receiver Command

For controllers that support the generation of event messages, it is required to implement the Set Event Receiver and Get Event Receiver commands. An event message is initiated by the satellite controller when a significant event happens (such as a power supply failure) and that event needs to be communicated to the system as soon as possible, without having to wait to respond to a command. The purpose of the Set Event Receiver command is to communicate to the satellite controller the slave address and LUN of the event receiver. When an event occurs, the satellite controller send the event messages to the slave address and LUN contained in the Set Event Receiver command. The command detail is given in Table 4.4, with the IPMB format of this command with an example is given in Figure 4.4.

**Table 4-4 Set Event Receiver Command**

Request Data	1	Event Receiver Slave Address. 0FFh disables Event Message Generation, Otherwise: [7:1] - IPMB (I 2 C) Slave Address [0] - always 0b when [7:1] hold I 2 C slave address
	2	[7:2] - reserved [1:0] - Event Receiver LUN
Response Data	1	Completion Code

#### 4.5 Get Event Receiver Command

This command is used to retrieve the present setting for the Event Receiver Slave Address and LUN. The command detail is given in Table 4-5, with the IPMB format of this command with an example is given in Figure 4.5.

**Table 4-5 Get Event Receiver Command**

Request Data	-	-
Response Data	1	Completion Code
	2	Event Receiver Slave Address. 0FFh indicates Event Message Generation has been disabled. Otherwise: [7:1] IPMB (I 2 C) Slave Address [0] always 0b when [7:1] hold I 2 C slave address
	3	[7:2] - reserved [1:0] - Event Receiver LUN

**Figure 4-4 Set Event Receiver Command**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 04h</b> (S/E request)				<b>rsLUN = 00</b>				10h		
byte3	<b>checksum = 9E h</b>								9Eh		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 00h</b> (Set Event Receiver)								00h		
byte7	<b>byte 1 = 20h</b> (Event Receiver address)								20h		
byte8	<b>byte 2 = 00h</b> (Event Receiver LUN)								00h		
byte9	<b>checksum = BCh</b>								BCh		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>				14h		
byte 3	<b>checksum = 88h</b>								88h		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 00h</b> (Set Event Receiver)								00h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>checksum = AAh</b>								AAh		

**Figure 4.5 Get Event Receiver Command**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 04h</b> (S/E Request)				<b>rsLUN = 00</b>				10h		
byte3	<b>checksum = 9Eh</b>								9Eh		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 01h</b> (Get Event Receiver)								01h		
byte7	<b>Checksum = DBh</b>								DBh		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>				1Ch		
byte 3	<b>checksum = C4h</b>								C4h		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 01h</b> (get Event Receiver)								01h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>data byte 2 = 20h</b> (Event Receiver Address)								20h		
byte 9	<b>data byte 3 = 00h</b> (Event Receiver LUN)								00h		
byte 10	<b>checksum = 89h</b>								89h		

## 4.6 Set Sensor Hysteresis

This command allows the hysteresis to be programmed on each sensor. The programmed hysteresis is added to a default hysteresis of 1, per the IPMI specification. A detailed example of the implementation of this command is given in Figure 4.6 and Table 4.6.

**Figure 4.6 - Set Sensor Hysteresis:**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rsSA = 60h</b>								52h		
byte 2	<b>netFn = 04h</b> (S/E request)				<b>rsLUN = 00</b>				10h		
byte 3	<b>checksum = 9E h</b>								9Eh		
byte 4	<b>rqSA = 20h</b>								20h		
byte 5	<b>rqSeq = 01h</b>				<b>rqLUN = 00</b>				04h		
byte 6	<b>cmd = 24h</b> (Set Sensor Hysteresis)								24h		
byte 7	<b>data byte 1 = 00h</b> (Sensor Number)								00h		
byte 8	<b>data byte 2 = FFh</b> (Reserved for future use)								FFh		
byte 9	<b>data byte 3 = 03h</b> (Positive-going Hysteresis)								03h		
byte 10	<b>data byte 4 = 03h</b> (Negative-going Hysteresis)								03h		
byte 11	<b>checksum = B3h</b>								B3h		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>								20h		
byte2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>				14h		
byte3	<b>checksum = CCh</b>								CCh		
byte4	<b>rsSA = 52h</b>								52h		
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>				04h		
byte6	<b>cmd = 24h</b> (Set Sensor Hysteresis)								24h		
byte7	<b>byte 1 = 00h</b> (Completion code - OK)								00h		
byte8	<b>checksum = 86h</b>								86h		

Request Data	1	sensor number (FFh = reserved)
	2	reserved for future 'hysteresis mask' definition. Write as 'FFh'
	3	Positive-going Threshold Hysteresis Value. Set to 00h if sensor does not support positive-going threshold hysteresis. This value is <i>added</i> to negative going thresholds to set the point where the asserted status for that threshold clears. Set to 00h if sensor does not support negative-going threshold hysteresis. See section 29.13.2, Hysteresis and Event Status and section 29.13.3, High-going versus Low-going Threshold Events.
	4	Negative-going Threshold Hysteresis Value. This value is <i>added</i> to negative going thresholds to set the point where the asserted status for that threshold clears. Set to 00h if sensor does not support negative-going threshold hysteresis.
Response Data	1	Completion Code

**Table 4-6 Set Sensor Hysteresis**

#### 4.7 Get Sensor Hysteresis

The Get Sensor Hysteresis command allows the requester to get the hysteresis settings for the sensor number contained in the request. A detailed example of a Get Sensor Hysteresis request and response is given in figure 4-7. The detail of the fields is given in table 4-7.

Request Data	1	sensor number (FFh = reserved)
	2	reserved for future 'hysteresis mask' definition. Write as 'FFh'
Response Data	1	Completion Code
	2	Positive-going Threshold Hysteresis Value. 00h if n/a.
	3	Negative-going Threshold Hysteresis Value. 00h if n/a.

**Table 4-7 Get Sensor Hysteresis**

## Command - Get Sensor Hysteresis:

Request:

Bits		
7 6 5 4 3 2 1 0		
byte1	<b>rsSA = 52h</b>	52h
byte2	<b>NetFn = 04h</b> (S/E request) <b>rsLUN = 00</b>	10h
byte3	<b>checksum = 9E h</b>	9Eh
byte4	<b>rqSA = 20h</b>	20h
byte5	<b>rqSeq = 01</b> <b>rqLUN = 00</b>	04h
byte6	<b>cmd = 25h</b> (Get Sensor Hysteresis)	25h
byte7	<b>byte 1 = 00h</b> (Sensor Number)	00h
byte8	<b>byte 2 = FFh</b> (Reserved for future use)	FFh
byte9	<b>checksum = B8h</b>	B8h

Response:

Bits		
7 6 5 4 3 2 1 0		
byte 1	<b>rqSA = 20h</b>	20h
byte 2	<b>netFn = 05h</b> (S/E response) <b>rqLUN = 00</b>	14h
byte 3	<b>checksum = CCh</b>	CCh
byte 4	<b>rsSA = 52h</b>	52h
byte 5	<b>rqSeq = 01h</b> <b>rsLUN = 00</b>	04h
byte 6	<b>cmd = 25h</b> (Get Sensor Hysteresis)	25h
byte 7	<b>data byte 1 = 00h</b> (Completion code - OK)	00h
byte 8	<b>data byte 3 = 03h</b> (Positive going Thresh Hyst)	03h
byte 9	<b>data byte 4 = 03h</b> (Negative going Thresh Hyst)	03h
byte 10	<b>checksum = 7Fh</b>	7Fh

**Figure 4-7 Get Sensor Hysteresis**

### 4.8 Set Sensor Thresholds Command

This command sets the thresholds that will be used to generate event messages (if enabled), on a sensor by sensor basis. The detail of the command is given in table 4.8, and an example of the request and response is given in figure 4.8.

**Figure 4.8 Set Sensor Thresholds:**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rsSA = 52h</b>										52h
byte 2	<b>netFn = 04h</b> (S/E request)					<b>rsLUN = 00</b>					10h
byte 3	<b>checksum = 9Eh</b>										9Eh
byte 4	<b>rqSA = 20h</b>										20h
byte 5	<b>rqSeq = 01h</b>					<b>rqLUN = 00</b>					04h
byte 6	<b>cmd = 26h</b> (Set Sensor Thresholds)										26h
byte 7	<b>data byte 1 = 00h</b> (Sensor Number)										00h
byte 8	<b>data byte 2 = 3Fh</b> (Threshold set enables)										3Fh
byte 9	<b>data byte 3 = B3h</b> (Lower non-critical Threshold)										B3h
byte 10	<b>data byte 4 = AAh</b> (Lower critical Threshold)										AAh
byte 11	<b>data byte 5 = A0h</b> (Lower non-recover Threshold)										A0h
byte 12	<b>data byte 6 = C6h</b> (Upper non-critical Threshold)										C6h
byte 13	<b>data byte 7 = CFh</b> (Upper critical Threshold)										CFh
byte 14	<b>data byte 8 = D9h</b> (Upper non-recover Threshold)										D9h
byte 15	<b>checksum = 0Ch</b>										0Ch

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>										20h
byte 2	<b>netFn = 05h</b> (S/E response)					<b>rqLUN = 00</b>					14h
byte 3	<b>checksum = C4h</b>										CCh
byte 4	<b>rsSA = 52h</b>										52h
byte 5	<b>rqSeq = 01h</b>					<b>rsLUN = 00</b>					04h
byte 6	<b>cmd = 26h</b> (Set Sensor Thresholds)										26h
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')										00h
byte 8	<b>checksum = 76h</b>										84h

Request Data	1	sensor number (FFh = reserved)
	2	[7:6] - reserved. Write as 00b. [5] - 1b = set upper non-recoverable threshold [4] - 1b = set upper critical threshold [3] - 1b = set upper non-critical threshold [2] - 1b = set lower non-recoverable threshold [1] - 1b = set lower critical threshold [0] - 1b = set lower non-critical threshold
	3	lower non-critical threshold. Ignored if bit 0 of byte 2 = 0
	4	lower critical threshold. Ignored if bit 1 of byte 2 = 0
	5	lower non-recoverable threshold. Ignored if bit 2 of byte 2 = 0
	6	upper non-critical threshold. Ignored if bit 3 of byte 2 = 0
	7	upper critical threshold value. Ignored if bit 4 of byte 2 = 0
	8	upper non-recoverable threshold value. Ignored if bit 5 of byte 2 = 0
Response Data	1	Completion Code

**Table 4-8 Set Sensor Thresholds**

#### 4.9 Get Sensor Thresholds Command

This command allows the requester to get the present threshold settings for the addressed sensor. The details of the command are given in table 4-9, and an example of the IPMB request and response is shown in figure 4-9.

Request Data	1	sensor number (FFh = reserved)
	1	Completion Code
Response Data	2	[7:6] - reserved. Write as 00b. Readable thresholds: This bit mask indicates which thresholds are readable. [5] - 1b = set upper non-recoverable threshold [4] - 1b = set upper critical threshold [3] - 1b = set upper non-critical threshold [2] - 1b = set lower non-recoverable threshold [1] - 1b = set lower critical threshold [0] - 1b = set lower non-critical threshold
	3	lower non-critical threshold. (if present, ignore on read otherwise)
	4	lower critical threshold. (if present, ignore on read otherwise)
	5	lower non-recoverable threshold. (if present, ignore on read otherwise)
	6	upper non-critical threshold. (if present, ignore on read otherwise)
	7	upper critical threshold value. (if present, ignore on read otherwise)
	8	upper non-recoverable threshold value. (if present, ignore on read otherwise)

**Table 4-9 Get Sensor Thresholds**

## Command – Get Sensor Thresholds:

Request:

Bits		
7 6 5 4 3 2 1 0		
byte 1	<b>rsSA = 52h</b>	52h
byte 2	<b>netFn = 04h</b> (S/E request) <b>rsLUN = 00</b>	10h
byte 3	<b>checksum = 9Eh</b>	9Eh
byte 4	<b>rqSA = 20h</b>	20h
byte 5	<b>rqSeq = 01h</b> <b>rqLUN = 00</b>	04h
byte 6	<b>cmd = 27h</b> (Get Sensor Thresholds)	27h
byte 7	<b>data byte 1 = 00h</b> (Sensor Number)	00h
byte 8	<b>checksum = B5h</b>	B5h

Response:

Bits		
7 6 5 4 3 2 1 0		
byte1	<b>rqSA = 20h</b>	20h
byte2	<b>netFn = 05h</b> (S/E response) <b>rqLUN = 00</b>	14h
byte3	<b>checksum = CCh</b>	CCh
byte4	<b>rsSA = 52h</b>	52h
byte5	<b>rqSeq = 01</b> <b>rsLUN = 00</b>	04h
byte6	<b>cmd = 27h</b> (Get Sensor Thresholds)	27h
byte7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')	00h
byte8	<b>byte 2 = 3Fh</b> (Threshold read enables)	3Fh
byte9	<b>byte 3 = B3h</b> (Lower non-critical Threshold)	B3h
byte10	<b>byte 4 = AAh</b> (Lower critical Threshold)	AAh
byte11	<b>byte 5 = A0h</b> (Lower non-recover Threshold)	A0h
byte12	<b>byte 6 = C6h</b> (Upper non-critical Threshold)	C6h
byte13	<b>byte 7 = CFh</b> (Upper critical Threshold)	CFh
byte 14	<b>byte 8 = D9h</b> (Upper non-recover Threshold)	D9H
byte 15	<b>checksum = D9h</b>	D9h

**Figure 4-9 Get Sensor Thresholds**

## 4.10 Set Sensor Event Enables

This command provides the ability to disable or enable Event Message Generation for individual sensor events.

A typical sensor will come up with Event Messages (EvM) enabled for all thresholds/states. Note that internal event flags and scanning will continue even though Event Message generation is disabled.

The details of the command are given in table 4.10, and an example of the IPMB request and response is shown in figure 4.10.

**Table 4-10 Set Sensor Event Enables**

Request Data	1	sensor number (FFh = reserved)
	2	<p>[7] - 0b = disable all Event Messages from this sensor (optional) [does not impact individual enable/disable status]          [6] - 0b = disable scanning on this sensor (optional)          [5:4] - 00b = do not change individual enables              01b = enable selected event messages              10b = disable selected event messages              11b = reserved          [3:0] – reserved</p>
	3	<p>For sensors with threshold based events:          [7] - 1b = select assertion event for upper non-critical going high          [6] - 1b = select assertion event for upper non-critical going low          [5] - 1b = select assertion event for lower non-recoverable going high          [4] - 1b = select assertion event for lower non-recoverable going low          [3] - 1b = select assertion event for lower critical going high          [2] - 1b = select assertion event for lower critical going low          [1] - 1b = select assertion event for lower non-critical going high          [0] - 1b = select assertion event for lower non-critical going low          For sensors with discrete events:          [7] - 1b = select assertion event for state bit 7          [6] - 1b = select assertion event for state bit 6          [5] - 1b = select assertion event for state bit 5          [4] - 1b = select assertion event for state bit 4          [3] - 1b = select assertion event for state bit 3          [2] - 1b = select assertion event for state bit 2          [1] - 1b = select assertion event for state bit 1          [0] - 1b = select assertion event for state bit 0</p>
	4	<p>For sensors with threshold based events:          [7:4] - reserved. Write as 0000b.          [3] - 1b = select assertion event for upper non-recoverable going high          [2] - 1b = select assertion event for upper non-recoverable going low          [1] - 1b = select assertion event for upper critical going high          [0] - 1b = select assertion event for upper critical going low          For sensors with discrete events:          [00h otherwise]          [7] - reserved. Write as 0b.          [6] - 1b = select assertion event for state bit 14          [5] - 1b = select assertion event for state bit 13          [4] - 1b = select assertion event for state bit 12          [3] - 1b = select assertion event for state bit 11          [2] - 1b = select assertion event for state bit 10          [1] - 1b = select assertion event for state bit 9          [0] - 1b = select assertion event for state bit 8</p>

5	<p>For sensors with threshold based events:</p> <p>[7] - 1b = select deassertion event for upper non-critical going high  [6] - 1b = select deassertion event for upper non-critical going low  [5] - 1b = select deassertion event for lower non-recoverable going high  [4] - 1b = select deassertion event for lower non-recoverable going low  [3] - 1b = select deassertion event for lower critical going high  [2] - 1b = select deassertion event for lower critical going low  [1] - 1b = select deassertion event for lower non-critical going high  [0] - 1b = select deassertion event for lower non-critical going low</p> <p>For sensors with discrete events:  (00h otherwise)</p> <p>[7] - 1b = select deassertion event for state bit 7  [6] - 1b = select deassertion event for state bit 6  [5] - 1b = select deassertion event for state bit 5  [4] - 1b = select deassertion event for state bit 4  [3] - 1b = select deassertion event for state bit 3  [2] - 1b = select deassertion event for state bit 2  [1] - 1b = select deassertion event for state bit 1  [0] - 1b = select deassertion event for state bit 0</p>
6	<p>For sensors with threshold based events:</p> <p>[7:4] - reserved. Write as 0000b.</p> <p>[3] - 1b = select deassertion event for upper non-recoverable going high  [2] - 1b = select deassertion event for upper non-recoverable going low  [1] - 1b = select deassertion event for upper critical going high  [0] - 1b = select deassertion event for upper critical going low</p> <p>For sensors with discrete events:  (00h otherwise)</p> <p>[7] - reserved. Write as 0b.</p> <p>[6] - 1b = select deassertion event for state bit 14  [5] - 1b = select deassertion event for state bit 13  [4] - 1b = select deassertion event for state bit 12  [3] - 1b = select deassertion event for state bit 11  [2] - 1b = select deassertion event for state bit 10  [1] - 1b = select deassertion event for state bit 9  [0] - 1b = select deassertion event for state bit 8</p>
1	Completion Code

Response Data

**Figure 4-10 Set Sensor Event Enables**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rsSA = 52h</b>								52h		
byte 2	<b>netFn = 04h</b> (S/E request)				<b>rsLUN = 00</b>				10h		
byte 3	<b>checksum = 9Eh</b>								9Eh		
byte 4	<b>rqSA = 20h</b>								20h		
byte 5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte 6	<b>cmd = 28h</b> (Set Sensor Event Enables)								28h		
byte 7	<b>byte 1 = 00h</b> (Sensor Number)								00h		
byte 8	<b>byte 2 = 10h</b> (Global enables)								10h		
byte 9	<b>byte 3 = 95h</b> (Event assertion enables-1 )								95h		
byte 10	<b>byte 4 = 0Ah</b> (Event assertion enables-2)								0Ah		
byte 11	<b>byte 5 = 95h</b> (Event deassertion enables-1)								95h		
byte 12	<b>byte 2 = 0Ah</b> (Event deassertion enables-2)								0Ah		
byte 13	<b>checksum = A6h</b>								A6h		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>				14h		
byte 3	<b>checksum = CCh</b>								CCh		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 28h</b> (Set Sensor Event Enables)								28h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>checksum = 82 h</b>								82h		

**Table 4-10 Set Sensor Event Enables**

## 4.11 Get Sensor Event Enables

This command returns the enabled/disabled state for Event Message Generation from the selected sensor. The details of the command are given in table 4-11, and an example of the IPMB request and response is shown in figure 4-11.

**Figure 4-11 Get Sensor Event Enables**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rsSA = 52h</b>										52h
byte 2	<b>netFn = 04h</b> (S/E request)				<b>rsLUN = 00</b>						10h
byte 3	<b>checksum = 9Eh</b>										9Eh
byte 4	<b>rqSA = 20h</b>										20h
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>						04h
byte 6	<b>cmd = 29h</b> (Get Sensor Event Enables)										29h
byte 7	<b>data byte 1 = 00h</b> (Sensor Number)										00h
byte 8	<b>checksum = B3 h</b>										B3h

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>										20h
byte2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>						14h
byte3	<b>checksum = CCh</b>										CCh
byte4	<b>rsSA = 52h</b>										52h
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>						04h
byte6	<b>cmd = 29h</b> (Get Sensor Event Enables)										29h
byte7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')										00h
byte8	<b>byte 2 = D0h</b> (Global enables)										D0h
byte9	<b>byte 3 = 95h</b> (Event assertion enables-1)										95h
byte10	<b>byte 4 = 0Ah</b> (Event assertion enables-2)										0Ah
byte11	<b>byte 5 = 95h</b> (Event deassertion enables-1)										95h
byte12	<b>byte 6 = 0Ah</b> (Event deassertion enables-2)										0Ah
byte 13	<b>checksum = 73h</b>										73h

**Table 4-11 Get Sensor Event Enables**

Request Data  
Response Data

1	sensor number (FFh = reserved)
1	Completion Code
2	[7] - 0b = All Event Messages disabled from this sensor [6] - 0b = Sensor scanning disabled [5:0] - reserved. Ignore on read.
3	For sensors with threshold based events: [7] - 1b = assertion event for upper non-critical going high enabled [6] - 1b = assertion event for upper non-critical going low enabled [5] - 1b = assertion event for lower non-recoverable going high enabled [4] - 1b = assertion event for lower non-recoverable going low enabled [3] - 1b = assertion event for lower critical going high enabled [2] - 1b = assertion event for lower critical going low enabled [1] - 1b = assertion event for lower non-critical going high enabled [0] - 1b = assertion event for lower non-critical going low enabled For sensors with discrete events: [7] - 1b = assertion event msg. for state bit 7 enabled [6] - 1b = assertion event msg. for state bit 6 enabled [5] - 1b = assertion event msg. for state bit 5 enabled [4] - 1b = assertion event msg. for state bit 4 enabled [3] - 1b = assertion event msg. for state bit 3 enabled [2] - 1b = assertion event msg. for state bit 2 enabled [1] - 1b = assertion event msg. for state bit 1 enabled [0] - 1b = assertion event msg. for state bit 0 enabled
4	For sensors with threshold based events: [7:4] - reserved. Write as 0000b. [3] - 1b = assertion event for upper non-recoverable going high enabled [2] - 1b = assertion event for upper non-recoverable going low enabled [1] - 1b = assertion event for upper critical going high enabled [0] - 1b = assertion event for upper critical going low enabled For sensors with discrete events: (00h otherwise) [7] - reserved. [6] - 1b = assertion event msg. for state bit 14 enabled [5] - 1b = assertion event msg. for state bit 13 enabled [4] - 1b = assertion event msg. for state bit 12 enabled [3] - 1b = assertion event msg. for state bit 11 enabled [2] - 1b = assertion event msg. for state bit 10 enabled [1] - 1b = assertion event msg. for state bit 9 enabled [0] - 1b = assertion event msg. for state bit 8 enabled
5	For sensors with threshold based events: [7] - 1b = deassertion event for upper non-critical going high enabled [6] - 1b = deassertion event for upper non-critical going low enabled [5] - 1b = deassertion event for lower non-recoverable going high enabled [4] - 1b = deassertion event for lower non-recoverable going low enabled [3] - 1b = deassertion event for lower critical going high enabled [2] - 1b = deassertion event for lower critical going low enabled [1] - 1b = deassertion event for lower non-critical going high enabled [0] - 1b = deassertion event for lower non-critical going low enabled For sensors with discrete events: [7] - 1b = deassertion event msg. for state bit 7 enabled [6] - 1b = deassertion event msg. for state bit 6 enabled [5] - 1b = deassertion event msg. for state bit 5 enabled [4] - 1b = deassertion event msg. for state bit 4 enabled [3] - 1b = deassertion event msg. for state bit 3 enabled [2] - 1b = deassertion event msg. for state bit 2 enabled [1] - 1b = deassertion event msg. for state bit 1 enabled [0] - 1b = deassertion event msg. for state bit 0 enabled

6	<p>For sensors with threshold based events:  [7:4] - reserved. Write as 0000b.  [3] - 1b = deassertion event for upper non-recoverable going high enabled  [2] - 1b = deassertion event for upper non-recoverable going low enabled  [1] - 1b = deassertion event for upper critical going high enabled  [0] - 1b = deassertion event for upper critical going low enabled</p> <p>For sensors with discrete events:  (00h otherwise)  [7] - reserved.  [6] - 1b = deassertion event msg. for state bit 14 enabled  [5] - 1b = deassertion event msg. for state bit 13 enabled  [4] - 1b = deassertion event msg. for state bit 12 enabled  [3] - 1b = deassertion event msg. for state bit 11 enabled  [2] - 1b = deassertion event msg. for state bit 10 enabled  [1] - 1b = deassertion event msg. for state bit 9 enabled  [0] - 1b = deassertion event msg. for state bit 8 enabled</p>
---	---

## 4.12 Get Sensor Event Status

The Get Sensor Event Status command is provided to support systems where sensor polling is used in addition to, or instead of, Event Messages for event detection. The Get Sensor Event Status is also the only way to get the 'latched' status for sensors that require manual re-arming of their event detection mechanism.

A device that implements a sensor must only generate a single Event Message for a given sensor event. (Retries may cause this message to be sent multiple times - but it is still the same message from an event handling point-of-view).

In order to track the fact that the event message has been sent, an implementation will typically implement an internal flag to indicate that the event condition has been met and the event generated. An 'auto- re-arm' sensor will clear its internal flag when the event condition goes away. A manual re-arm sensor requires a *Re-arm Sensor Events* command to clear the flag in order for event generation to be re-enabled for the event. The *Get Sensor Event Status* commands may be considered as returning the state of these internal flags.

Since the 'Event Status' for a manual re-arm sensor stays until manual cleared, the state is sometime referred to as the 'Event History' or just 'History' for the sensor. The event status gets updated when the controller detects a *state change* or transition between the present state and the previous state (conditioned by hysteresis as appropriate). The exception to this is when a sensor is re-armed by a *Re-arm Sensor* or *Set Event Receiver* command. In this case, the event status gets updated after the controller gets its first reading for the sensor.

If the sensor is 'auto- re-arm' then the command returns unlatched present event status for the sensor. The event status for auto- re-arm sensors can be derived from the present status information returned in a Get Sensor Reading command, if the hysteresis values are known. For this reason, the Get Sensor Event Status

command is typically not implemented for auto- re-arm sensors. Instead, if system management software needs to determine event status, it derives it from the *Get Sensor Reading* and hysteresis settings. The format of the *Get Sensor Event Status* response is dependent on whether the sensor was threshold based or discrete. For an auto re-arm, threshold based sensor, the present threshold comparison is the event status. This is redundant to the threshold comparison status returned with the 'Get Sensor Reading' command if the sensor has no hysteresis. Otherwise, software can derive the event status from the Get Sensor Reading command if it knows the hysteresis value.

The sensors in this application are all of the 'auto-re-arm' type.

The details of the command are given in table 4-12, and an example of the IPMB request and response is shown in figure 4-12.

**Table 4-12 Get Sensor Event Status**

Request Data	1	sensor number (FFh = reserved)
Response Data	1	Completion Code
	2	<p>[7] - 0b = All Event Messages disabled from this sensor          [6] - 0b = Sensor scanning disabled          [5] - 1b = initial update in progress. This bit is set to indicate that a 're-arm' or 'Set Event Receiver' command has been used to request an update of the sensor status, and that update has not occurred yet. Software should use this bit to avoid getting an incorrect status while the first sensor update is in progress. This bit is only required if it is possible for the controller to receive and process a 'Get Sensor Reading' or 'Get Sensor Event Status' command for the sensor before the update has completed. This is most likely to be the case for sensors, such as fan RPM sensors, that may require seconds to accumulate the first reading after a re-arm.          [4:0] - reserved. Ignore on read.</p>
	3	<p>For sensors with threshold based events:          (High-going events are asserted when value first becomes =threshold. Low-going events are asserted when value first becomes =corresponding threshold.)          [7] - 1b = assertion event condition for upper non-critical going high occurred          [6] - 1b = assertion event condition for upper non-critical going low occurred          [5] - 1b = assertion event condition for lower non-recoverable going high occurred          [4] - 1b = assertion event condition for lower non-recoverable going low occurred          [3] - 1b = assertion event condition for lower critical going high occurred          [2] - 1b = assertion event condition for lower critical going low occurred          [1] - 1b = assertion event condition for lower non-critical going high occurred          [0] - 1b = assertion event condition for lower non-critical going low occurred          For sensors with discrete events:          [7] - 1b = state 7 assertion event occurred          [6] - 1b = state 6 assertion event occurred          [5] - 1b = state 5 assertion event occurred          [4] - 1b = state 4 assertion event occurred          [3] - 1b = state 3 assertion event occurred          [2] - 1b = state 2 assertion event occurred          [1] - 1b = state 1 assertion event occurred          [0] - 1b = state 0 assertion event occurred</p>

**Table 4-12 Get Sensor Event Status (continued)**

4*	<p>For sensors with threshold based events:            [7:4] - reserved. Write as 0000b.            [3] - 1b = assertion event condition for upper non-recoverable going high occurred            [2] - 1b = assertion event condition for upper non-recoverable going low occurred            [1] - 1b = assertion event condition for upper critical going high occurred            [0] - 1b = assertion event condition for upper critical going low occurred            For sensors with discrete events:            (00h otherwise)            [7] - reserved. Ignore on read.            [6] - 1b = state 14 assertion event occurred            [5] - 1b = state 13 assertion event occurred            [4] - 1b = state 12 assertion event occurred            [3] - 1b = state 11 assertion event occurred            [2] - 1b = state 10 assertion event occurred            [1] - 1b = state 9 assertion event occurred            [0] - 1b = state 8 assertion event occurred</p>
5*	<p>For sensors with threshold based events:            (High-going events are deasserted when value goes less than the corresponding threshold minus the positive-going hysteresis value. Low-going events are deasserted when value goes greater than the corresponding threshold plus the negative-going hysteresis value.)            [7] - 1b = deassertion event condition for upper non-critical going high occurred            [6] - 1b = deassertion event condition for upper non-critical going low occurred            [5] - 1b = deassertion event condition for lower non-recoverable going high occurred            [4] - 1b = deassertion event condition for lower non-recoverable going low occurred            [3] - 1b = deassertion event condition for lower critical going high occurred            [2] - 1b = deassertion event condition for lower critical going low occurred            [1] - 1b = deassertion event condition for lower non-critical going high occurred            [0] - 1b = deassertion event condition for lower non-critical going low occurred            For sensors with discrete events:            [7] - 1b = state 7 deassertion event occurred            [6] - 1b = state 6 deassertion event occurred            [5] - 1b = state 5 deassertion event occurred            [4] - 1b = state 4 deassertion event occurred            [3] - 1b = state 3 deassertion event occurred            [2] - 1b = state 2 deassertion event occurred            [1] - 1b = state 1 deassertion event occurred            [0] - 1b = state 0 deassertion event occurred</p>
6*	<p>For sensors with threshold based events:            [7:4] - reserved. Write as 0000b.            [3] - 1b = deassertion event condition for upper non-recoverable going high occurred            [2] - 1b = deassertion event condition for upper non-recoverable going low occurred            [1] - 1b = deassertion event condition for upper critical going high occurred            [0] - 1b = deassertion event condition for upper critical going low occurred            For sensors with discrete events:            (0h otherwise)            [7] - reserved. Ignore on read.            [6] - 1b = state 14 deassertion event occurred            [5] - 1b = state 13 deassertion event occurred            [4] - 1b = state 12 deassertion event occurred            [3] - 1b = state 11 deassertion event occurred            [2] - 1b = state 10 deassertion event occurred            [1] - 1b = state 9 deassertion event occurred            [0] - 1b = state 8 deassertion event occurred</p>

\* = Devices must accept a variable number of response data bytes (3 to 6). This requirement is to allow a reduction in the number of data bytes that must be transferred. It is recommended that implementations only return the number of data bytes required to satisfy the command.

**Figure 4-12 Get Sensor Event Status**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>										52h
byte2	<b>netFn = 04h</b> (S/E request)				<b>rsLUN = 00</b>						10h
byte3	<b>checksum = 9Eh</b>										9Eh
byte4	<b>rqSA = 20h</b>										20h
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>						04h
byte6	<b>cmd = 2Bh</b> (Get Sensor Event Status)										2Bh
byte7	<b>byte 1 = 00h</b> (Sensor Number)										00h
byte8	<b>checksum = B1h</b>										B1h

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>										20h
byte 2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>						1Ch
byte 3	<b>checksum = C4h</b>										C4h
byte 4	<b>rsSA = 52h</b>										52h
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>						04h
byte 6	<b>cmd = 2Bh</b> (Get Sensor Event Status)										2Bh
byte 7	<b>data byte 1 = 00h</b> (Completion Code = OK)										00h
byte 8	<b>data byte 2 = 00h</b> (event info)										00h
byte 9	<b>data byte 3 = 00h</b> (Event Assertion Status Data a)										00h
byte 10	<b>data byte 4 = 00h</b> (Event Assertion Status Data b)										00h
byte 11	<b>data byte 5 = 00h</b> (Event De-Assertion Status Data a)										00h
byte 12	<b>Data byte 6 = 00h</b> (Event De-Assertion Status Data b)										00h
byte 13	<b>checksum = 7FF h</b>										7Fh

## 4.13 Get Sensor Reading Command

This command returns the present reading for sensor. The sensor device may return a stored version of a periodically updated reading, or the sensor device may scan to obtain the reading after receiving the request. The meaning of the state bits returned by Discrete sensors is based on the Event/Reading Type code from the SDR for the sensor. This command returns the present reading for sensor. The sensor device returns a stored version of a periodically updated reading. The meaning of the state bits returned by Discrete sensors is based on the Event/Reading Type code from the SDR for the sensor. The command detail is given in Table 4-13, with the IPMB format of this command with an example is given in Figure 4-13.

**Table 4-13 Get Sensor Reading Command**

Request Data	1	sensor number (FFh = reserved)
Response Data	1	Completion Code.
	2	Sensor reading Byte 1: byte of reading. Ignore on read if sensor does not return an numeric (analog) reading.
	3	[7] - 0b = All Event Messages disabled from this sensor [6] - 0b = sensor scanning disabled [5] - 1b = initial update in progress. This bit is set to indicate that a 're-arm' or 'Set Event Receiver' command has been used to request an update of the sensor status, and that update has not occurred yet. Software should use this bit to avoid getting an incorrect status while the first sensor update is in progress. This bit is only required if it is possible for the controller to receive and process a 'Get Sensor Reading' or 'Get Sensor Event Status' command for the sensor before the update has Completed. This is most likely to be the case for sensors, such as fan RPM sensors, that may require seconds to accumulate the first reading after a re-arm. [4:0] – reserved. Ignore on read.
	(4)	For threshold-based sensors Present threshold comparison status [7:6] – reserved. Returned as 1b. Ignore on read. [5] - 1b = at or above ( ) upper non-recoverable threshold [4] - 1b = at or above ( ) upper critical threshold [3] - 1b = at or above ( ) upper non-critical threshold [2] - 1b = at or below ( ) lower non-recoverable threshold [1] - 1b = at or below ( ) lower critical threshold [0] - 1b = at or below ( ) lower non-critical threshold For discrete reading sensors [7] - 1b = state 7 asserted [6] - 1b = state 6 asserted [5] - 1b = state 5 asserted [4] - 1b = state 4 asserted [3] - 1b = state 3 asserted [2] - 1b = state 2 asserted [1] - 1b = state 1 asserted [0] - 1b = state 0 asserted
	(5)	For discrete reading sensors only. (Optional) (00h Otherwise) [7] – reserved. Returned as 1b. Ignore on read. [6] - 1b = state 14 asserted [5] - 1b = state 13 asserted [4] - 1b = state 12 asserted [3] - 1b = state 11 asserted [2] - 1b = state 10 asserted [1] - 1b = state 9 asserted [0] - 1b = state 8 asserted

**Figure 4-13 Get Sensor Readings Command**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>										52h
byte2	<b>netFn = 04h</b> (S/E request)				<b>rsLUN = 00</b>						10h
byte3	<b>checksum = 9Eh</b>										9Eh
byte4	<b>rqSA = 20h</b>										20h
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>						04h
byte6	<b>cmd = 2Dh</b> (Get Sensor Reading)										2Dh
byte7	<b>byte 1 = 00h</b> (Sensor Number)										00h
byte8	<b>checksum = AFh</b>										AFh

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>										20h
byte 2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>						14h
byte 3	<b>checksum = CCh</b>										CCh
byte 4	<b>rsSA = 52h</b>										52h
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>						04h
byte 6	<b>cmd = 2Dh</b> (Get Sensor Reading)										2Dh
byte 7	<b>data byte 1 = 00h</b> (Completion Code = OK)										00h
byte 8	<b>data byte 2 = BBh</b> (Sensor Reading)										BBh
byte 9	<b>data byte 3 = 00h</b> (Sensor Status)										00h
byte 10	<b>data byte 4 = C0h</b> (Sensor Data 1)										C0h
byte 11	<b>data byte 5 = 00h</b> (Sensor Data 2)										00h
byte 12	<b>checksum = 02h</b>										02h

#### 4.14 Event Message

Event Messages can be generated for alert conditions (such as a failure of one output from a power supply). These are typically generated when a sensor reading exceeds a set threshold. If that threshold is enabled for event messages, an event message will be generated. The IPMB format of this command with an example is given in Figure 4-14. The command format is given in Table 4-14, the

Event Message Fields are shown in table 4-15, and the Event Request Message Event Data Field Contents are detailed in Table 4-16. The Sensor and Event Code Tables are listed in Appendix B.

**Figure 4.14 Event Message**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 20h</b>										20h
byte2	<b>netFn = 04h</b> (S/E request)				<b>rsLUN = 00</b>						10h
byte3	<b>checksum = D0h</b>										D0h
byte4	<b>rqSA = 52h</b>										52h
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>						04h
byte6	<b>cmd = 02h</b> (Event Message)										02h
byte7	<b>byte 1 = 04h</b> (EvMRev=V1.5)										04h
byte8	<b>byte 2 = 02h (voltage)</b> (Sensor type)										02h
byte9	<b>byte 3 = 00h</b> (Sensor Number)										00h
byte10	<b>byte 4 = 00h (assertion)</b> (Event direction/Type)										01h
byte11	<b>byte 5 = 05h (digital)</b> (Event Data1)										50h
byte12	<b>byte 2 = 01h</b> (Event Data2)										B7h
byte13	<b>byte 2 = 01h</b> (Event Data3)										B7h
byte 14	<b>checksum = h</b>										16h

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 52h</b>										52h
byte 2	<b>netFn = 05h</b> (S/E response)				<b>rqLUN = 00</b>						14h
byte 3	<b>checksum = 9Ah</b>										9Ah
byte 4	<b>rsSA = 20h</b>										20h
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>						04h
byte 6	<b>cmd = 02h</b> (Event Message)										02h
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')										00h
byte 8	<b>checksum = DAh</b>										DAh

**Table 4-14 Event Message Format**

	byte	data field
Request Data	1	EvMRev
	2	Sensor Type
	3	Sensor #
	4	Event Dir   Event Type
	5	Event Data 1
	6	Event Data 2
	7	Event Data 3
Response Data	1	Completion Code.

**Table 4-15 Event Message Fields**

Field	Description
EvMRev	One byte. Event Message Revision. This field is used to identify different revisions of the Event Message format. <i>The revision number shall be 04h for Event Messages that comply with the format given in this specification.</i> IPMI v1.0 messages use 03h. It is recommended that software be able to interpret both versions.
Sensor Type	One byte. Indicates the event class or type of sensor that generated the Event Message. The Sensor Type Codes are specified in Appendix B, Sensor and Event Code Tables.
Sensor #	One byte. A unique number (within a given sensor device) representing the 'sensor' within the management controller that generated the Event Message. Sensor numbers are used for both identification and access of sensor information, such as getting and setting sensor thresholds.
Event Dir	1-bit. Indicates the event transition direction. (0 = Assertion Event, 1 = De-assertion Event)
Event Type	7-bits. This field indicates the type of threshold crossing or state transition (trigger) that produced the event. This is encoded using the Event/Reading Type Code. See <i>Appendix C-1, Sensor and Event Code Tables.</i>
Event Data	One to three Bytes. The remainder of the Event Message data according to the class of the Event Type for the sensor (threshold, discrete, or OEM). The contents and format of this field is specified in Table 4-16, Event Request Message Event Data Field Contents, below.

**Table 4-16 Event Request Message Event Data Field Contents**

Sensor Class	Event Data
threshold	<p><u>Event Data 1</u>            [7:6] - 00b = unspecified byte 2            01b = trigger reading in byte 2            10b = OEM code in byte 2            11b = sensor-specific event extension code in byte 2            [5:4] - 00b = unspecified byte 3            01b = trigger threshold value in byte 3            10b = OEM code in byte 3            11b = sensor-specific event extension code in byte 3            [3:0] - Offset from Event/Reading Code for threshold event.  <u>Event Data 2</u> reading that triggered event, FFh or not present if unspecified.  <u>Event Data 3</u> threshold value that triggered event, FFh or not present if unspecified. If present, byte 2 must be present.</p>
discrete	<p><u>Event Data 1</u>            [7:6] - 00b = unspecified byte 2            01b = previous state and/or severity in byte 2            10b = OEM code in byte 2            11b = sensor-specific event extension code in byte 2            [5:4] - 00b = unspecified byte 3            01b = reserved            10b = OEM code in byte 3            11b = sensor-specific event extension code in byte 3            [3:0] - Offset from Event/Reading Code for discrete event state  <u>Event Data 2</u>            [7:4] - Optional offset from 'Severity' Event/Reading Code. (0Fh if unspecified).            [3:0] - Optional offset from Event/Reading Type Code for previous discrete event state. (0Fh if unspecified.)  <u>Event Data 3</u> Optional OEM code. FFh or not present if unspecified.</p>
OEM	<p><u>Event Data 1</u>            [7:6] - 00b = unspecified in byte 2            01b = previous state and/or severity in byte 2            10b = OEM code in byte 2            11b = reserved            [5:4] - 00b = unspecified byte 3            01b = reserved            10b = OEM code in byte 3            11b = reserved            [3:0] - Offset from Event/Reading Type Code  <u>Event Data 2</u>            [7:4] - Optional OEM code bits or offset from 'Severity' Event/Reading Type Code. (0Fh if unspecified).            [3:0] - Optional OEM code or offset from Event/Reading Type Code for previous event state. (0Fh if unspecified).  <u>Event Data 3</u> Optional OEM code. FFh or not present or unspecified.</p>

Last modified: 2-10-04

## 5. Firmware Commands

In the IPMI specification V1.5, for the Firmware Network Function, there is not a specific command set detailed. The firmware request and response format is the same as the application messages, but the type and content are considered device specific. The following command set is defined and optimized for this application using a Philips P89C66x family processor. The commands should be executed in the same order as presented below. The Program Flash command will be repeated until the firmware transfer is complete. The Exit Firmware Update command would then follow. If any other command is issued before the Start Firmware Update command request is issued, a “Command Not Supported” (C1h) error code will be reported in the response.

### 5.1 Start Firmware Update

The Start Firmware Update command transfers control of the IPMB (I<sup>2</sup>C) interrupt driven interface in the application program to a polled IPMB (I<sup>2</sup>C) interface in the update code that is “ROMed” into a portion of memory that does not get erased. This update code operates without interrupts enabled during the firmware update process. This is because the memory space vectored to upon receiving enabled interrupts is part of the flash memory space that is being updated. The command detail for the Start Firmware Update command is given in Table 5-1, with an example of the IPMB format of the command in Figure 5-1.

**Table 5-1 Start Firmware Update Command**

Request Data	-	-
Response Data	1	Completion Code

**Figure 5-1 Start Firmware Update Command**

**Request:**

		Bits									
		7	6	5	4	3	2	1	0		
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 08h</b> (f/w request)				<b>rsLUN = 00</b>				20h		
byte3	<b>checksum = 80Ah</b>								8Eh		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 01h</b> (Start f/w Update)								01h		
byte7	<b>Checksum = DBh</b>								DBh		

**Response:**

		Bits									
		7	6	5	4	3	2	1	0		
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 09h</b> (f/w response)				<b>rqLUN = 00</b>				24h		
byte 3	<b>checksum = C4h</b>								BCh		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 01h</b> (Start f/w Update)								01h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>checksum = A9h</b>								A9h		

## 5.2 Erase Flash

The Erase Flash command erases the flash memory space occupied by the program code being updated. The main program, all functions, interrupt vectors, start-up code and look-up tables are contained in this space and will require re-programming after this command is executed. The response time to this command will exceed the normal specified response time in the IPMB specification due to the time it takes to complete an erase cycle. However, the specification does make an exception for controller specific application and firmware commands as long as the exception is detailed in the specification. The time delay for a valid response to this command can be up to 30 seconds. The command detail for the Erase Flash command is given in Table 5-2, with an example of the IPMB format of the command in Figure 5-2.

**Table 5-2 Erase Flash Command**

Request Data  
Response Data

-	-
1	Completion Code

**Figure 5-2 Erase Flash Command**

**Request:**

		Bits								
		7	6	5	4	3	2	1	0	
byte1	<b>rsSA = 52h</b>								52h	
byte2	<b>netFn = 08h</b> (f/w request)				<b>rsLUN = 00</b>				20h	
byte3	<b>checksum = 8Eh</b>								8Eh	
byte4	<b>rqSA = 20h</b>									
byte4	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				20h	
byte5	<b>cmd = 08h</b> (erase flash)								04h	
byte6	<b>Checksum = D4h</b>								08h	
byte7									D4h	

**Response:**

		Bits								
		7	6	5	4	3	2	1	0	
byte 1	<b>rqSA = 20h</b>								20h	
byte 2	<b>netFn = 09h</b> (f/w response)				<b>rqLUN = 00</b>				24h	
byte 3	<b>checksum = C4h</b>								BCh	
byte 4	<b>rsSA = 52h</b>								52h	
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h	
byte 6	<b>cmd = 08h</b> (erase flash)								08h	
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h	
byte 8	<b>checksum = A2h</b>								A2h	

### 5.3 Program Flash

The Program Flash command uses the basic Intel Hex Object File Format with the leading colon and trailing checksum stripped off each record. This is typically an ASCII file. For this application, this is converted to hexadecimal and embedded into the IPMB format. Theoretically, the byte count of a single record can be 0-255, however, most assemblers, compilers, and programmers deal with 16 bytes per record. This application will use the 16 byte (maximum) per record format since that also fits well within the specified IPMB maximum of 32 bytes per transfer (16 bytes of program data plus 11 bytes of overhead = 27 bytes). If an 'FFh' completion code is returned, this is an indication of a programming failure. ( For example, if an attempt is made to program a location of code memory that is not erased.) The command detail for the Program Flash command is given in Table 5-3, with an example of the IPMB format of the command in Figure 5-3.

**Figure 5-3 Program Flash Command**

**Request:**

		Bits									
		7	6	5	4	3	2	1	0		
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>NetFn = 08h</b> (f/w request)				<b>rsLUN = 00</b>				20h		
byte3	<b>checksum = 8Eh</b>								8Eh		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 09h</b> (program flash)								09h		
byte7	<b>Byte Count=n</b> (10h=max)								10h		
byte8	<b>Load Addr of first Byte (MSB)</b>								01h		
byte9	<b>Load Addr of first Byte (LSB)</b>								00h		
byte10	<b>Record type = 00h</b> (00h =data, 01h=end)								00h		
byte n+10	<b>0:n data bytes to program</b>								xxh		
byte n+11	<b>Checksum</b>								xxh		

**Response:**

		Bits									
		7	6	5	4	3	2	1	0		
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 09h</b> (f/w response)				<b>rqLUN = 00</b>				24h		
byte 3	<b>checksum = C4h</b>								BCh		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 09h</b> (program flash)								09h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>checksum = A1h</b>								A1h		

Request Data	1	Byte count = n = 10h (max)
	2	Load address, MS Byte
	3	Load address, LS Byte
	4	Record type, 00h = Data Record, 01h = End Record
	4+n	Data bytes to program (16 max)
Response Data	1	Completion Code

**Table 5-3 Program Flash Command**

#### 5.4 Exit Firmware Update

The Exit Firmware Update command transfers control of the IPMB (I<sup>2</sup>C) interface back to the main program that has now been updated. A response with a satisfactory completion code indicates that the transfer is complete and the updated software is ready to receive and respond to commands, and, to generate enabled event messages. The response time to this command will exceed the normal specified response time in the IPMB specification due to the time it takes to complete an erase cycle. However, the specification does make an exception for controller specific application and firmware commands as long as the exception is detailed in the specification. The time delay for a valid response to this command can be up to 5 seconds. The command detail for the Program Flash command is given in Table 5-4, with an example of the IPMB format of the command in Figure 5-4.

**Figure 5-4 Exit Firmware Update Mode Command**

**Request:**

		Bits								
		7	6	5	4	3	2	1	0	
byte1	<b>rsSA = 52h</b>								52h	
byte2	<b>netFn = 08h</b> (f/w request)				<b>rsLUN = 00</b>				20h	
byte3	<b>checksum = 8Eh</b>								8Eh	
byte4	<b>rqSA = 20h</b>									
byte4	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				20h	
byte5	<b>cmd = 04h</b> (exit f/w update)								04h	
byte6	<b>Checksum = D8h</b>								04h	
byte7									D8h	

**Response:**

		Bits								
		7	6	5	4	3	2	1	0	
byte 1	<b>rqSA = 20h</b>								20h	
byte 2	<b>netFn = 09h</b> (f/w response)				<b>rqLUN = 00</b>				24h	
byte 3	<b>checksum = C4h</b>								BCh	
byte 4	<b>rsSA = 52h</b>								52h	
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h	
byte 6	<b>cmd = 04h</b> (exit f/w update)								04h	
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h	
byte 8	<b>checksum = A6h</b>								A6h	

**Table 5-4 Exit Firmware Update Mode Command**

Request Data	-	-
Response Data	1	Completion Code

Last Update: 2-10-04

## 6.0 FRU Device Commands

The following sections describe the FRU (Field Replaceable Unit) Inventory Device format, access commands, and a content example. The FRU Inventory data contains information such as the serial number, part number, asset tag, and short descriptive string for the FRU. The contents of a FRU Inventory Record are specified in the *Platform Management FRU Information Storage Definition*. The sections of that specification that are applicable are summarized in Appendix G.

This FRU Inventory Device is a 'logical' device. It is not implemented as a separate physical device, but is contained in the Flash Memory of the Satellite Controller.

### 6.1 Get FRU Inventory Area Info Command

Returns the overall the size of the FRU Inventory Area in this device, in bytes.

**Table 6-1, Get FRU Inventory Area Info Command**

	byte	data field
Request Data	1	FRU Device ID. FFh = reserved.
Response Data	1	Completion Code
	2	FRU Inventory area size in bytes, LS Byte
	3	FRU Inventory area size in bytes, MS Byte
	4	[7:1] - reserved [0] 0b = Device is accessed by bytes, 1b = Device is accessed by words

**Figure 6.1 Get FRU Inventory Area Info Command**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 0Ah</b> (Storage Request)				<b>rsLUN = 00</b>				28h		
byte3	<b>checksum = 86h</b>								86h		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 10h</b> (Get FRU Inventory Area)								10h		
byte7	<b>FRU Device ID= 00h</b> (FRU Device ID)								00h		
byte8	<b>Checksum = CCh</b>								CCh		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rqSA = 20h</b>								20h		
byte 2	<b>netFn = 0Bh</b> (Storage response)				<b>rqLUN = 00</b>				2Ch		
byte 3	<b>checksum = B4h</b>								B4h		
byte 4	<b>rsSA = 52h</b>								52h		
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h		
byte 6	<b>cmd = 10h</b> (Get FRU Inventory Area)								10h		
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h		
byte 8	<b>data byte 2 = C8h</b> (FRU Area - LS byte)								C8h		
byte 9	<b>data byte 3 = 00h</b> (FRU Area - MS byte)								00h		
byte 10	<b>data byte 4 = 00h</b> (FRU byte access)								00h		
byte 11	<b>checksum = E0h</b>								E0h		

## 6.2 Read FRU Data Command

The command returns the specified data from the FRU Inventory Info area. This is effectively a 'low level' direct interface to a non-volatile storage area. This 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 does not correspond to the physical address used in the device that provides the non-volatile storage. For example, FRU information could be kept in FLASH at physical address 1234h, however offset 0000h would still be used with this command to access the start of the FRU information. IPMI FRU device data (devices that are formatted per [FRU]) as well as processor and DIMM FRU data always starts from offset 0000h unless otherwise noted.

Note that while the offsets are 16-bit values, allowing FRU devices of up to 64K words, the *count to read*, *count returned*, and *count written* fields are only 8-bits. This is in recognition of the limitations on the sizes of messages. For example, as of this writing, IPMB messages are limited to 32-bytes total.

**Table 6-2, Read FRU Data Command**

	byte	data field
Request Data	1	FRU Device ID. FFh = reserved.
	2	FRU Inventory Offset to read, LS Byte
	3	FRU Inventory Offset to read, MS Byte Offset is in bytes or words per device access type returned in the <i>Get FRU Inventory Area Info</i> command.
Response Data	4	Count to read --- count is '1' based
	1	Completion Code
	2	Count returned --- count is '1' based
	3:2+n	Requested data

**Figure 6-2, Read FRU Data Command**

**Request:**

		Bits								
		7 6 5 4 3 2 1 0								
byte 1	<b>rsSA = 52h</b>								52h	
byte 2	<b>netFn = 0Ah</b> (Storage request)				<b>rsLUN = 00</b>				28h	
byte 3	<b>checksum = 86h</b>								86h	
byte 4	<b>rqSA = 20h</b>								20h	
byte 5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h	
byte 6	<b>cmd = 11h</b> (Read FRU Data)								11h	
byte 7	<b>byte 1 = 00h</b> (FRU Device ID)								00h	
byte 8	<b>byte 2 = 00h</b> (FRU Inventory to read – LS byte)								00h	
byte 9	<b>byte 3 = 00h</b> (FRU Inventory to read – MS byte)								00h	
byte 10	<b>byte 4 = 00h</b> (Count to Read)								00h	
byte 11	<b>checksum = AAh</b>								00h	

**Response:**

		Bits								
		7 6 5 4 3 2 1 0								
byte 1	<b>rqSA = 20h</b>								20h	
byte 2	<b>netFn = 0Bh</b> (Storage response)				<b>rqLUN = 00</b>				2Ch	
byte 3	<b>checksum = 8Eh</b>								8Eh	
byte 4	<b>rsSA = 52h</b>								52h	
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h	
byte 6	<b>cmd = 11h</b> (Read FRU Data)								11h	
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h	
byte 8	<b>data byte 2 = 01h</b> (Count Returned)								01h	
byte 9	<b>data byte 3:2+n =</b> (Requested data from FRU)								xxh	
byte 10	<b>checksum = xx h</b>								xxh	

### 6.3 Write FRU Data Command

The command writes the specified byte or word to the FRU Inventory Info area. This is a 'low level' direct interface to a non-volatile storage area. This means that the interface does not interpret or check any semantics or formatting for the

data being written. The offset used in this command is a 'logical' offset that does not correspond to the physical address used in device that provides the non-volatile storage. For example, FRU information could be kept in FLASH at physical address 1234h, however offset 0000h would still be used with this command to access the start of the FRU information. IPMI FRU device data (devices that are formatted per [FRU]) as well as processor and DIMM FRU data always starts from offset 0000h unless otherwise noted. Updating the FRU Inventory Data is presumed to be a system level, privileged operation. There is no requirement for devices implementing this command to provide mechanisms for rolling back the FRU Inventory Area in the case of incomplete or incorrect writes.

**Table 6-3, Write FRU Data Command**

	byte	data field
Request Data	1	FRU Device ID. FFh = reserved.
	2	FRU Inventory Offset to write, LS Byte
	3	FRU Inventory Offset to write, MS Byte
Response Data	4:3+n	Data to write
	1	Completion Code
	2	Count written --- count is '1' based

**Figure 6-3 Write FRU Command**

**Request:**

		Bits							
		7	6	5	4	3	2	1	0
byte1	<b>RsSA = 52h</b>								52h
byte2	<b>netFn = 0Ah</b> (Storage request)				<b>rsLUN = 00</b>				28h
byte3	<b>checksum = 86h</b>								86h
byte4	<b>rqSA = 20h</b>								20h
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h
byte6	<b>cmd = 12h</b> (Write FRU Data)								12h
byte7	<b>FRU Device ID=00h</b>								00h
byte8	FRU offset to write – LS byte								01h
byte9	FRU offset to write – MS byte								00h
byte10	3+n Bytes to Write								xxh
byte n+11	Checksum								xxh

**Response:**

		Bits							
		7	6	5	4	3	2	1	0
byte 1	<b>rqSA = 20h</b>								20h
byte 2	<b>netFn = 0Bh</b> (Storage response)				<b>rqLUN = 00</b>				2Ch
byte 3	<b>checksum = 8Eh</b>								8Eh
byte 4	<b>rsSA = 52h</b>								52h
byte 5	<b>rqSeq = 01h</b>				<b>rsLUN = 00</b>				04h
byte 6	<b>cmd = 12h</b> (Write FRU Data)								12h
byte 7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')								00h
byte 8	<b>data byte 2 = xxh</b> (Count Written)								xxh
byte 9	<b>checksum = xxh</b>								xxh

## 6.4 FRU Format and Example Content:

The FRU memory holds one mandatory and typically one or more optional fields. The mandatory field is the Common Header field and is 8 Bytes in length and always located at offset 0000h. For this application the Product info area will also be included as it contains information such as Serial Number, Model Number, etc., for the product. The specification on the format of these fields is given in Appendix G. The FRU will be supplied blank, to be programmed later. This includes the field checksums in the locations defined here and in Appendix G.

The Format and example values for the Common Header are given below in Table 6.4

**Table 6.4 - Common Header**

field length	offset	field	value
1	0	Common Header Format Version 7:4 - reserved, write as 0000b 3:0 - format version number = 1h for this specification.	01h
1	1	Internal Use Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.	00h
1	2	Chassis Info Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.	00h
1	3	Board Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.	00h
1	4	Product Info Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.	01h
1	5	MultiRecord Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.	10h
1	6	PAD, write as 00h	00h
1	7	Common Header Checksum (zero checksum)	EEh

The Product information area can be of different lengths depending on the area allocated for each of the information fields. The format with example values and area allocations for the Product Information Area is given in Table 6.5.

**Table 6.5 - Product Info Area**

field length	off-set	field	value
1	08h	Product Area Format Version 7:4 - reserved, write as 0000b 3:0 - format version number = 1h for this specification	01h
1	09h	Product Area Length (in multiples of 8 bytes) – 120 bytes	0Fh
1	0Ah	Language Code ('0' or '25 decimal' for English )	00h
1	0Bh	Manufacturer Name type/length byte	E0h
32	0Ch	Manufacturer Name bytes	programmable
1	2Ch	Product Name type/length byte	programmable
16	2Dh	Product Name bytes	programmable
1	3Dh	Product Part/Model Number type/length byte	Program D0h
16	3Eh	Product Part/Model Number bytes	programmable
1	4Eh	Product Version type/length byte	Program C8H
8	4Fh	Product Version bytes	programmable
1	57h	Product Serial Number type/length byte*	Program D0h
16	58h	Product Serial Number bytes*	programmable
1	68h	Asset Tag type/length byte	Program D0h
16	69h	Asset Tag	programmable
1	79h	FRU File ID type/length byte	00h
1	7Ah	FRU File ID bytes. The FRU File version field is a pre-defined Field provided as a manufacturing aid for verifying the file that was used during manufacture or field update to load the FRU information. The content is manufacturer-specific. This field is also provided in the Board Info area. Either or both fields may be 'null'.	00h
0	-	Custom product info area fields, if any (must be preceded with type/length byte)	-
1	7Bh	C1h (type/length byte encoded to indicate no more info fields).	C1h
3	7Ch	00h - any remaining unused space	00h,00h,00h
1	7Fh	Product Info Area Checksum (zero checksum)	xxh

\* These fields are always encoded as if the Language Code were English. i.e. if the type/length code bits 7:6=11b the serial number will always be interpreted as ASCII+Latin 1, not UNICODE.

The following presents the specification of the type/length byte.

7:6 - type code

00 - binary or unspecified

01 - BCD plus

10 - 6-bit ASCII, packed (overrides Language Codes)

11 - Interpretation depends on Language Codes. 11b indicates 8-bit ASCII + Latin 1 if the Language Code is English

5:0 - number of data bytes.

000000 indicates that the field is empty. When the type code is 11b, a length of

000001 indicates 'end of fields'. I.e. Type/Length = C1h indicates 'end of fields'.

**Table 6-6 DC Output Records (Record Type 0x01)**

Output 1:

offset	Field Length	field	Decimal Value / Notes	Hex Value
80h	1	Record Type ID	1	0x01
81h	1	7:7 – End of list 6:4 – Reserved, write as 000b 3:0 – Record Format version (=2h unless otherwise specified)	2	0x02
82h	1	Record Length	18 bytes	0x12
83h	1	Record Checksum (zero checksum)		
84h	1	Header Checksum (zero checksum)		
85h	1	Output information 7:7 – Standby 6:4 – Reserved, write as 000b 3:0 – Output number	Output #1	0x01
86h	2	Nominal voltage (10 mV)	5.00V	0x01F4
88h	2	Maximum negative voltage deviation (10 mV)	4.85V	0x01E5
8Ah	2	Maximum positive voltage deviation (10 mV)	5.25V	0x020D
8Ch	2	Ripple and Noise pk-pk 10Hz to 30 MHz (mV)	50mV	0x0032
8Eh	2	Minimum current draw (mA)	0	0x0000
90h	2	Maximum current draw (mA)	30.000A	0x7530

Output 2:

offset	Field Length	field	Decimal Value / Notes	Hex Value
92h	1	Record Type ID	1	0x01
93h	1	7:7 – End of list 6:4 – Reserved, write as 000b 3:0 – Record Format version (=2h unless otherwise specified)	2	0x02
94h	1	Record Length		
95h	1	Record Checksum (zero checksum)		
96h	1	Header Checksum (zero checksum)		
97h	1	Output information 7:7 – Standby 6:4 – Reserved, write as 000b 3:0 – Output number	Output #2	0x02
98h	2	Nominal voltage (10 mV)	3.30V	0x014A
9Ah	2	Maximum negative voltage deviation (10 mV)	3.20V	0x0140
9Ch	2	Maximum positive voltage deviation (10 mV)	3.46V	0x015A
9Eh	2	Ripple and Noise pk-pk 10Hz to 30 MHz (mV)	50Mv	0x0032
A0h	2	Minimum current draw (mA)	0 mA	0x0000
A2h	2	Maximum current draw (mA)	40.000 A	0x9C40

Output 3:

offset	field length	field	Decimal Value / Notes	Hex Value
A4h	1	Record Type ID	1	0x01
A5h	1	7:7 – End of list 6:4 – Reserved, write as 000b 3:0 – Record Format version (=2h unless otherwise specified)	2	0x02
A6h	1	Record Length	18 Bytes	
A7h	1	Record Checksum (zero checksum)		
A8h	1	Header Checksum (zero checksum)		
A9h	1	Output information 7:7 – Standby 6:4 – Reserved, write as 000b 3:0 – Output number	Output #3	0x03
AAh	2	Nominal voltage (10 mV)	12.00V	0x04B0
ACh	2	Maximum negative voltage deviation (10 mV)	11.40V	0x0474
A Eh	2	Maximum positive voltage deviation (10 mV)	12.60V	0x04EC
B0h	2	Ripple and Noise pk-pk 10Hz to 30 MHz (mV)	240mV	0x00F0
B2h	2	Minimum current draw (mA)	0 A	0x0000
B4h	2	Maximum current draw (mA)	5.000 A	0x1388

Output 4:

offset	field length	field	Decimal Value / Notes	Hex Value
B6h	1	Record Type ID	1	0x01
B7h	1	7:7 – End of list 6:4 – Reserved, write as 000b 3:0 – Record Format version (=2h unless otherwise specified)	2	0x82
B8h	1	Record Length	18 Bytes	0x12
B9h	1	Record Checksum (zero checksum)		0xEF
BAh	1	Header Checksum (zero checksum)		0x82
BBh	1	Output information 7:7 – Standby 6:4 – Reserved, write as 000b 3:0 – Output number	Output #4	0x04
BCh	2	Nominal voltage (10 mV)	-12.00V	0xFB50
BEh	2	Maximum negative voltage deviation (10 mV)	-11.40V	0XFB8C
C0h	2	Maximum positive voltage deviation (10 mV)	-12.60V	0XFB14
C2h	2	Ripple and Noise pk-pk 10Hz to 30 MHz (mV)	240 mV	0x00F0
C4h	2	Minimum current draw (mA)	0 A	0x0000
C6h	2	Maximum current draw (mA)	-1.00A	0XFC18

Last Modified 2-8-04

## 7.1 OEM Commands

The OEM commands are used to implement device specific functions not specified in the IPMI/IPMB specifications. This includes commands for monitoring and overriding the state of the Power OK LED and the power supply shutdown output, a device status command, device implemented FRU type commands, and device set-up commands. A special class of OEM commands, Vendor Only OEM Commands, was also specified to allow the vendor to program information such as device ID information, serial numbers, and firmware creation dates, during the manufacturing process.

### 7.1 General OEM Commands

#### 7.1.1 Set Fault LED State OEM command

This command allows the normal state of the Fault (FAL) LED to be over-written. This is an amber LED that indicates one of the outputs, or a thermal sensor, has exceeded an alarm threshold when 'ON'. Normal operation is indicated by the amber 'FAL' LED being in the 'OFF' state. . The IPMB format of this command with an example is given in Figure 7.1.1. The command detail is given in Table 7.1.1.

**Table 7.1.1 Set Fault LED State**

Request Data	1	LED number , ( 01h for "Fault (FAL)" LED)
	2	LED state: 0 = off, 1 = on, FFh = auto (Fault Status – Power Supply Good = LED is off)
Response Data	1	Completion code

**Figure 7.1.1 Set Fault LED State**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>										52h
byte2	<b>netFn = 30h</b> (OEM request)				<b>rsLUN = 00</b>						C0h
byte3	<b>checksum = EEh</b>										EEh
byte4	<b>rqSA = 20h</b>										20h
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>						04h
byte6	<b>cmd = 50h</b> (Set LED State)										50h
byte7	<b>byte 1 = 01h</b> (LED Number, 01h=fault LED)										01h
byte8	<b>byte 2 = 01h</b> (LED State: 0=off, 1=on, FF=auto)										01h
byte9	<b>checksum =8Ah</b>										8Ah

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>										20h
byte2	<b>netFn = 31h</b> (OEM response)				<b>rqLUN = 00</b>						C4h
byte3	<b>checksum = 1Ch</b>										1Ch
byte4	<b>rsSA = 52h</b>										52h
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>						04h
byte6	<b>cmd = 50h</b> (Set LED State)										50h
byte7	<b>byte 1 = 00h</b> (Completion code - OK)										00h
byte8	<b>checksum =5Ah</b>										5Ah

### 7.1.2 Get Fault LED State OEM command

This command gets the state of the Fault (FAL) LED. Since the fault LED can be overridden, the system may want to determine if that LED is in the override state. The IPMB format of this command with an example is given in Figure 7.1.2. The command detail is given in Table 7.1.2.

**Table 7.1.2 Get Fault LED State**

Request Data  
Response Data

1	LED number
1	Completion code
2	LED state: 0 = Forced off, 1 = Forced on, FF= Normal Operation

**Figure 7.1.2 Get Fault LED State**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 30h</b> (OEM request)				<b>rsLUN = 00</b>				C0h		
byte3	<b>checksum = EEh</b>								EEh		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 51h</b> (Get LED State)								51h		
byte7	<b>byte 1 = 01h</b> (LED Number, 01h=fault LED)								01h		
byte8	<b>checksum = 8Ah</b>								8Ah		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>								20h		
byte2	<b>netFn = 31h</b> (OEM response)				<b>rqLUN = 00</b>				C4h		
byte3	<b>checksum = 1Ch</b>								1Ch		
byte4	<b>rsSA = 52h</b>								52h		
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>				04h		
byte6	<b>cmd = 51h</b> (Get LED State)								51h		
byte7	<b>byte 1 = 00h</b> (Completion code - OK)								00h		
byte8	<b>byte 2 = 01h</b> (LED State: 0=off, 1=on, FF=auto)								01h		
byte9	<b>checksum = 4Bh</b>								4Bh		

### 7.1.3 Set Inhibit State OEM command

This command allows the state of the power supply “Inhibit (INH\_OUT)” output, normally under hardware control via the EN and INH inputs, to be over-riden. Two integer parameters are specified. The first parameter, the inhibit delay, is specified in 10 ms increments and represents the time delay from when the command is received to when the outputs will be inhibited. The second parameter is the inhibit duration, specified in seconds. This parameter specifies the time duration that the inhibit signal is in the active state. If the parameter specified in the request is non-zero for the IPMI Inhibit duration parameter, the power supply will be inhibited for the duration of the timeout and then return to the “Outputs Enabled” state. If the timeout parameter is zero on the inhibit duration parameter, then the power supply will remain in the IPMI Inhibit state until an IPMI de-inhibit command is received. The timeout parameters are only used when the command is an inhibit command (inhibit state = 0) and is ignored when the command is a de-inhibit command (inhibit state = 1). The following table lists the possible states of the Inhibit Output, the actions that can occur in that state, and the result of those actions (transition to another state or remain in the same state). The microcontroller will be initialized to the Hardware inhibit state, but will typically transition quickly to the Outputs enabled state when EN is low (asserted) and INH is high (de-asserted). The green Input LED will be ‘ON’ in the “Outputs Enabled” state. If input power is present and the Inhibit state is either in the “Hardware Inhibit” state or the “IPMI Inhibit” state, then the green LED will blink at 1 second intervals.

state	action	result
Outputs Enabled	EN goes ‘High’	Transition to ‘Hardware Inhibit’ state
	INH goes ‘Low’	Transition to ‘Hardware Inhibit’ state
	IPMI Inhibit Command	Transition to ‘IPMI Inhibit’ state after inhibit delay
	IPMI de-Inhibit Command	Return response with error code – no change of state
Hardware Inhibit	EN goes ‘Low’(w/ INH high)	Transition to ‘Outputs Enabled’ state
	INH goes ‘High’(w/ EN Low)	Transition to ‘Outputs Enabled’ state
	IPMI Inhibit Command	Return response with error code – no change of state
	IPMI de-Inhibit Command	Return response with error code – no change of state
IPMI Inhibit	EN goes ‘Low’	Transition to ‘Hardware Inhibit’ state
	INH goes ‘High’	Transition to ‘Hardware Inhibit’ state
	IPMI de-Inhibit Command	Transition to ‘Outputs Enabled’ state
	IPMI Inhibit Command	Reset Timeout parameter – no change of state
	Timeout of Inhibit Command	Transition to ‘Outputs Enabled’ state

The IPMB format of this command with an example is given in Figure 7.1.3. The command detail is given in Table 7.1.3.

**Table 7.1.3 Set Inhibit State**

Request Data	1	Control Device number , 01h for Power Supply Inhibit control
	2	state: 0 = IPMI Inhibit, 1 = IPMI de-Inhibit
	3	Inhibit Delay – 10ms increments – ms byte
	4	Inhibit Delay – 10ms increments – 1s byte
	4	Inhibit Duration – seconds – ms byte

Response Data	6	Inhibit Duration – seconds – ls byte
	1	Completion code

**Figure 7.1.3 Set Inhibit State**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>								52h		
byte2	<b>netFn = 30h</b> (OEM request)				<b>rsLUN = 00</b>				C0h		
byte3	<b>checksum = h</b>								EEh		
byte4	<b>rqSA = 20h</b>								20h		
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>				04h		
byte6	<b>cmd = 20h</b> (Set Inhibit State)								20h		
byte7	<b>byte 1 = 01h</b> (Device Number)								01h		
byte8	<b>byte 2 = 01h</b> (Inhibit State: 0=assert, 1=de-assert)								01h		
byte9	<b>byte 3 = 00h</b> (Inhibit Delay- MS byte)								00h		
byte10	<b>byte 4 = 00h</b> (Inhibit Delay - LS byte)								00h		
byte11	<b>byte 5 = 00h</b> (Inhibit Duration- MS byte)								00h		
byte12	<b>byte 6 = 00h</b> (Inhibit Duration - LS byte)								00h		
byte13	<b>checksum =BAh</b>								BAh		

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>								20h		
byte2	<b>netFn = 31h</b> (OEM response)				<b>rqLUN = 00</b>				C4h		
byte3	<b>checksum = 1Ch</b>								1Ch		
byte4	<b>rsSA = 52h</b>								52h		
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>				04h		
byte6	<b>cmd = 20h</b> (Set Inhibit State)								20h		
byte7	<b>byte 1 = 00h</b> (Completion code - OK)								00h		
byte8	<b>checksum =8Ah</b>								8Ah		

#### 7.1.4 Get Inhibit State OEM command

This command allows the system to get the power supply inhibit state. The Power Supply Inhibit is initialized to the “Hardware Inhibit” state, but will typically transition to the “Outputs Enabled” state unless inhibited by assertion of the inhibit (INH) input or de-assertion of the enable (EN) input. Once in the Outputs Enabled state, the state can also transition to the “IPMI Inhibit” state via the “Set Inhibit State” command. The IPMB format of this command with an example is given in Figure 7.1.4. The command detail is given in Table 7.1.4.

**Table 7.1.4 Get Inhibit State**

Request Data	1	Control device number (01h, for power supply inhibit)
Response Data	1	Completion code
	2	Inhibit State state: 0 = Hardware Inhibit, 1 = Outputs Enabled, FFh = IPMI Inhibit

**Figure 7.1.4 Get Inhibit State**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rsSA = 52h</b>										52h
byte2	<b>netFn = 30h</b> (OEM request)				<b>rsLUN = 00</b>						C0h
byte3	<b>checksum = Eeh</b>										Eeh
byte4	<b>rqSA = 20h</b>										20h
byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>						04h
byte6	<b>cmd = 21h</b> (Get Inhibit State)										21h
byte7	<b>byte 1 = 01h</b> (Device Number)										01h
byte8	<b>checksum =BAh</b>										BAh

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>										20h
byte2	<b>netFn = 31h</b> (OEM response)				<b>rqLUN = 00</b>						C4h
byte3	<b>checksum = h</b>										E4h
byte4	<b>rsSA = 52h</b>										52h
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>						04h
byte6	<b>cmd = 21h</b> (Get Inhibit State)										21h
byte7	<b>byte 1 = 00h</b> (Completion code - OK)										00h
byte8	<b>byte 2 = 01h</b> (Inhibit State)										01h
byte9	<b>checksum =88h</b>										88h

### 7.1.5 Get Device Status OEM command

The “Get Device Status” OEM command provides a summary status of all the sensors monitored by the device. Normally, in the “Get Sensor Reading” response, the third byte after the completion code would contain the present threshold status of that sensor. The “Get Device Status” command provides the same status information as the status byte in the “Get Sensor Reading” response except all the sensors’ status bytes are provided in response to this command. The IPMB format of this command with an example is given in Figure 7.1.5. The command detail is given in Table 7.1.5.

**Table 7.1.5 Get Device Status**

Request Data	-	-
Response Data	1	Completion Code
	2	Status of Sensor 0
	3	Status of Sensor 1
	4	Status of Sensor 2
	5	Status of Sensor 3
	6	Status of Sensor 4
	7	Status of Sensor 5
	8	Status of Sensor 6
	9	Status of Sensor 7
	10	Status of Sensor 8

**Figure 7.1.5: Get Device Status Command**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
Byte 1	<b>rsSA = 52h</b>										52h
Byte2	<b>netFn = 30h</b> (OEM request)				<b>rsLUN = 00</b>						C0h
Byte3	<b>checksum = 0Ah</b>										E0h
Byte4	<b>rqSA = 20h</b>										20h
Byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>						04h
Byte6	<b>cmd = 04h</b> (get device status)										04h
Byte7	<b>checksum = D8h</b>										D8h

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>										20h
byte2	<b>netFn = 31h</b> (OEM response)				<b>rqLUN = 00</b>						14h
byte3	<b>checksum = CCh</b>										CCh
byte4	<b>rsSA = 60h</b>										60h
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>						04h
byte6	<b>cmd = 04h</b> (get device status)										04h
byte7	<b>data byte 1 = 00h</b> (completion code. 00h = 'OK')										00h
byte8	<b>byte 2 = 3Fh</b> (Sensor 0 status)										C0h
byte9	<b>byte 3 = C0h</b> (Sensor 1 status)										C0h
byte10	<b>byte 4 = C0h</b> (Sensor 2 status)										C0h
byte11	<b>byte 5 = C0h</b> (Sensor 3 status)										C0h
byte12	<b>byte 6 = C0h</b> (Sensor 4 status)										C0h
byte13	<b>byte 7 = C0h</b> (Sensor 5 status)										C0h
byte 14	<b>byte 8 = C0h</b> (Sensor 6 status)										C0h
byte 15	<b>byte 8 = C0h</b> (Sensor 7 status)										C0h
byte 16	<b>byte 8 = 00h</b> (Sensor 8 status)										00h
byte 17	<b>checksum = D8h</b>										D8h

## 7.2 OEM Special Commands – Initial Set-Up Commands

The OEM Special Commands are to be used for initial set-up of the product utilizing this IPMI/IPMB satellite controller. The commands are used to program the device ID information into non-volatile memory. This information can be read later by the application using the “Get Device ID” command described earlier in this document. The OEM Special Commands will use the Net Function code of 32h (also in the controller specific OEM group), to prevent confusion with the general OEM commands described earlier that use the Net Function code of 30h. An Additional OEM Special command supported is a lock programmed memory command. This allows the option of locking the memory where the device ID information is located from future programming attempts. Also included in this command is support to lock two different areas of the FRU inventory area. The manufacturer could lock one area with the serial number, model number, etc. for that product. The asset tag area could be left non-programmed, allowing the end user to program and lock this area separately.

### 7.2.1 Program Device ID information

The Program Device ID Information command stores the provided information into non-volatile memory so that it can be read later by the “Get Device ID” command described earlier in section 3.1. Refer to this section and appendix A for a detailed breakdown of the device ID information. The IPMB format of this command with an example is given in Figure 7.2.1. The command detail is given in Table 7.2.1.

**Figure 7.2.1. Program Device ID Information**

Request:

		Bits									
		7 6 5 4 3 2 1 0									
byte 1	<b>rsSA = 52h</b>								52h		
byte 2	<b>netFn = 32h</b> (OEM request)				<b>rsLUN = 00</b>				C8h		
byte 3	<b>checksum = E6h</b>								E6h		
byte 4	<b>rqSA = 20h</b>								20h		
byte 5	<b>rqSeq = 01h</b>				<b>rqLUN = 00</b>				04h		
byte 6	<b>cmd = 01h</b> (program device id)								01h		
byte 7	<b>data byte 2 = 00h</b> (device id)								00h		
byte 8	<b>data byte 3 = 81h</b> (device rev)								81h		
byte 9	<b>data byte 4 = 01h</b> (major f/w rev)								01h		
byte 10	<b>data byte 5 = 01h</b> (minor f/w rev)								01h		
byte 11	<b>data byte 7 = 21h</b> (device support. EG&sensors)								21h		
byte 12	<b>data byte 8 = xxh</b> (Manufacturer ID, LS byte)								69h		
byte 13	<b>data byte 9 = xxh</b> (Manufacturer ID, middle byte)								31h		
byte 14	<b>data byte 10 = xxh</b> (Manufacturer ID, MS byte)								01h		
byte 15	<b>data byte 11 = xxh</b> (product ID, LS byte)								86h		
byte 16	<b>data byte 12 = xxh</b> (Product ID, MS byte)								48h		
byte 17	<b>checksum = CEh</b>								CEh		

Response:

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>								20h		
byte2	<b>netFn = 33h</b> (OEM response)				<b>rqLUN = 00</b>				CCh		
byte3	<b>checksum = 14h</b>								14h		
byte4	<b>rsSA = 52h</b>								52h		
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>				04h		
byte6	<b>cmd = 01h</b> (program device id)								01h		
byte7	<b>byte 1 = 00h</b> (Completion code - OK)								00h		
byte8	<b>checksum = A9h</b>								A9h		

**Table 7.2.1 Program Device ID Information**

Request Data	1	Device ID. 00h = unspecified.
	2	Device Revision
	3	Firmware Revision 1
	4	Firmware Revision 2
	5	Additional Device Support
	6:8	Manufacturer ID, LS Byte first.
	9:10	Product ID, LS Byte first.
	Response Data	1

**7.2.2 Lock Vendor OEM Program Data**

The Lock Vendor OEM Program Data command is used to “lock” the information programmed by the Vendor Only OEM commands to prevent accidental corruption of the programmed data. Once this command has been executed once, and an error free response generated, any future attempts to program the area specified by the command will be unsuccessful. After this, for region 0, a program data command (i.e. Program Device ID Information) will be acknowledged, but not responded to. Also, in the IPMI FRU commands there is not a provision to lock any FRU regions. As a result, regions were defined in this command that allow the product information area not including the asset tag(region 1), the asset tag (region 2), and the FRU multi-record area (region 3) to be locked individually. An attempt to write to a locked FRU region would result in a response with an “Insufficient Privilege” completion code. Any reprogramming of locked regions would require a firmware update. The IPMB format of this command with an example is given in Figure 7.2.2. The command detail is given in Table 7.2.2

**Table 7.2.2 Lock Vendor OEM Program Data**

Request Data	1	Region to Lock, dev id=00h, product FRU = 01h, asset tag=02h, FRU multirecord = 03h
Response Data	1	Completion Code

**Figure7.2.2: Lock Vendor OEM Program Data**

**Request:**

		Bits									
		7 6 5 4 3 2 1 0									
Byte1	<b>rsSA = 52h</b>										52h
Byte2	<b>netFn = 32h</b> (VO-OEM request)				<b>rsLUN = 00</b>						C8h
Byte3	<b>checksum = E6h</b>										E6h
Byte4	<b>rqSA = 20h</b>										20h
Byte5	<b>rqSeq = 01</b>				<b>rqLUN = 00</b>						04h
Byte6	<b>cmd = 05 h</b> (lock program data)										05h
Byte6	<b>Region to Lock</b>										00h
Byte7	<b>checksum = D7h</b>										D7h

**Response:**

		Bits									
		7 6 5 4 3 2 1 0									
byte1	<b>rqSA = 20h</b>										20h
byte2	<b>netFn = 33h</b> (OEM response)				<b>rqLUN = 00</b>						CCh
byte3	<b>checksum = 14h</b>										14h
byte4	<b>rsSA = 52h</b>										52h
byte5	<b>rqSeq = 01</b>				<b>rsLUN = 00</b>						04h
byte6	<b>cmd = 05h</b> (lock program data)										05h
byte7	<b>byte 1 = 00h</b> (Completion code - OK)										00h
byte8	<b>checksum = A5h</b>										A5h

Last Modification: 2-26-04

## 8.0 Completion Codes

### 8.1 Error Handling and Supported Completion Codes

All Response Messages specified in this document include a *completion code* as the first byte in the data field of the response. A management controller that gets a request to an invalid (unimplemented) LUN must return an error completion code using that LUN as the responder's LUN (RsLUN) in the response. The completion code indicates whether the associate Request Message completed successfully and normally, and if not, provides a value indicating the completion condition.

Completion Codes work at the 'command' level. They are responses to the interpretation of the command *after* it has been received and validated through the messaging interface. Errors at the 'network' (messaging interface) level are handled with a different error reporting mechanism. For example, If the requester sends a message to the satellite controller with the wrong slave address, the satellite controller will not acknowledge the address. The exception to this is a broadcast message. In broadcast messages if the slave address is wrong, the first few bytes of the request will be acknowledged but the remaining bytes will not be acknowledged and no response will be generated. For a request that has an error in the first checksum. The request will be acknowledged but not responded to. For other errors, the satellite controller will respond with an Error Code. The error codes supported and the detail of the error cause are listed in Table 8-1.

**Table 8-1. Error Codes**

Code	Definition (from IPMI specification)	Notes
00h	Command Completed Normally.	OK
C1h	Invalid Command. Used to indicate an unrecognized or unsupported command.	The command received doesn't match a supported command
C2h	Command invalid for given LUN.	Not a valid LUN
C7h	Request data length invalid.	Number of bytes in the request didn't match what's expected for the command
C9h	Parameter Out of Range	Read or Write request includes a FRU address that is out of it's size range. This error code is also returned if a Read or Write request to the FRU contains a Device ID that is not the same as the ID returned when issuing the "Get Device ID" command.
CBh	Requested Sensor, data, or record not present.	The sensor number requested was out of the range of valid sensors
CCh	Invalid data field in Request	Error in checksum 2
D4h	Cannot Execute Command – Insufficient Privilege level	This is returned when trying to write to a protected area of the FRU
FFh	Unspecified Error	This is returned when trying to program an area of FRU that is not erased. This will also occur during a firmware update if there is a programming error or the flash area being programmed is not erased.

Last Modification: 2-10-04

## Appendix A – Command Detail - Get Device ID

The following presents additional specifications and descriptions for the Device ID response fields:

**Device ID/Device Instance** This number is specified by the manufacturer identified by the Manufacturer ID field. The Device ID field allows controller-specific software to identify the unique application command, OEM fields, and functionality that are provided by the controller. Controllers that have different application commands, or different definitions of OEM fields, are expected to have different Device ID Appendix values. Controllers that implement identical sets of applications commands can have the same Device ID in a given system. Thus, a 'standardized' controller could be produced where multiple instances of the controller are used in a system, and all have the same Device ID value. [The controllers would still be differentiable by their address, location, and associated information for the controllers in the Sensor Data Records.] The Device ID is typically used in combination with the Product ID field such that the Device IDs for different controllers are unique under a given Product ID. A controller can optionally use the Device ID as an 'instance' identifier if more than one controller of that kind is used in the system. Though implementing a Device GUID is the preferred method for uniquely identifying controllers. (See section 17.8, *Get Device GUID*) *This field is binary encoded.*

**Device Revision** The least significant nibble of the Device Revision field is used to identify when significant hardware changes have been made to the implementation of the management controller that cannot be covered with a single firmware release. That is, this field would be used to identify two builds off the same code firmware base, but for different board fab levels. For example, device revision "1" might be required for 'fab X and earlier' boards, while device revision "2" would be for 'fab Y and later' boards. *This field is binary encoded and unsigned.*

**Firmware Revision 1** Major Revision. 7-bits. This field holds the major revision of the firmware. This field shall be incremented on major changes or extensions of the functionality of the firmware - such as additions, deletions, or modifications of the command set. *This field is binary encoded and unsigned.* The Device Available bit is used to indicate whether normal command set operation is available from the device, or it is operating in a state where only a subset of the normal commands are available. This will typically be because the device is in a firmware update state. It may also indicate that full command functionality is not available because the device is in its initialization phase or an SDR update is in progress. Note that the revision information obtained when the Device Available bit is '1' shall be indicative of the code version that is *in effect*. Thus, the version information may vary with the Device Available bit state.

**Firmware Revision 2** Minor Revision. This field holds the minor revision of the firmware. This field will increment for minor changes such as bug fixes. *This field is BCD encoded.*

**IPMI Version** This field holds the version of the IPMI specification that the controller is compatible with. This indicates conformance with this document, including event message formats and mandatory command support. *This field is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits. The value shall be 51h indicating conformance with this specification, version 1.5.*

### **Additional Device Support**

This field indicates the logical device support that the device provides in addition to the IPM and Application logical devices.

**Manufacturer ID** This field uses the Internet Assigned Numbers Authority (<http://www.iana.org/>) SMI Network Management Private Enterprise Codes a.k.a. "Enterprise Numbers" for identifying

the manufacturer responsible for the specification of functionality of the vendor (OEM) -specific commands, codes, and interfaces used in the controller. For example, an event in the SEL could have OEM values in the event record. An application that parses the SEL could extract the controller address from the event record contents and use it to send the 'Get Device ID' command and retrieve the Manufacturer ID. A manufacturer-specific application could then do further interpretation based a-priori knowledge of the OEM field, while a generic cross-platform application would typically just use the ID to present the manufacturer's name alongside uninterpreted OEM event values. The manufacturer that defines the functionality is not necessarily the manufacturer that created the physical microcontroller. For example, Vendor A may create the controller, but it gets loaded with Vendor B's firmware. The Manufacture ID would be for Vendor B, since they're the party that defined the controller's functionality. The Manufacturer ID value from the *Get Device ID* command does not override Manufacturer or OEM ID fields that are explicitly defined as part of a command or record format. If no vendor-specific functionality is defined, it is recommended that the field can either be loaded with the Manufacturer ID of the party that is responsible for the firmware for the controller, or the value FFFFh to indicate 'unspecified'. *This field is binary encoded, and unsigned.*

**Product ID** This value can be used in combination with the Manufacturer ID and Device ID values to identify the product-specific element of the controller-specific functionality. This number is specified by the manufacturer identified by the Manufacturer ID field. Typically, a controller-specific application would use the Product ID to identify the type of board, module, or system that the controller is used in, instead of using the data from the FRU information associated with the controller.

#### **Auxiliary Firmware Revision Information**

This field is optional. If present, it holds additional information about the firmware revision, such as boot block or internal data structure version numbers. The meanings of the numbers are specific to the vendor identified by Manufacturer ID (see below). When the vendor-specific definition is not known, generic utilities should display each byte as 2-digit hexadecimal numbers, with byte 13 displayed first as the most-significant byte.

## Appendix B - Sensor Data Record Detail

### Sensor Data Record Type 01, Full Sensor Record ( Table 37-1 IPMI V1.5)

byte	Field Name	size	Description
SENSOR RECORD HEADER			
1	Record ID – L	2	The Record ID is used by the Sensor Data Repository device for record organization and access. It is <i>not</i> related to the sensor ID.
2	Record ID – H		
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. (51h for this specification) <i>BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.</i>
4	Record Type	1	Record Type Number = 01h, Full Sensor Record
5	Record Length	1	Number of remaining record bytes following.
RECORD KEY BYTES			
6	Sensor Owner ID	1	[7:1] - 7-bit I 2 C Slave Address, or 7-bit system software ID. [0] - 0b = ID is IPMB Slave Address, 1b = system software ID
7	Sensor Owner LUN	1	[7:4] - Channel Number The Channel Number can be used to specify access to sensors that are located on management controllers that are connected to the BMC via channels other than the primary IPMB. (Note: In IPMI v1.5 the ordering of bits 7:2 of this byte have changed to support the 4-bit channel number.) [3:2] - reserved [1:0] - Sensor Owner LUN. LUN in the Sensor Owner that is used to send/receive IPMB messages to access the sensor. 00b if system software is Sensor Owner.
8	Sensor Number	1	Unique number identifying the sensor behind a given slave address and LUN. Code FFh reserved.
RECORD BODY BYTES			
9	Entity ID	1	Indicates the physical entity that the sensor is monitoring or is otherwise associated. See Table 37-12, Entity ID Codes.
10	Entity Instance	1	[7] - 0b = treat entity as a physical entity per Entity ID table 1b = treat entity as a logical container entity. For example, if this bit is set, and the Entity ID is 'Processor', the container entity would be considered to represent a logical 'Processor Group' rather than a physical processor. This bit is typically used in conjunction with an Entity Association record. [6:0] - Instance number for entity. (See section 33.1, <i>System- and Device-relative Entity Instance Values</i> for more information) 00h-5Fh system-relative Entity Instance. The Entity Instance number must be unique for each different entity of the same type Entity ID in the system. 60h-7Fh device-relative Entity Instance. The Entity Instance number must only be unique relative to the management controller providing access to the Entity.
11	Sensor Initialization	1	[7] - reserved. Write as 0b. [6] - Init Scanning 1b = enable scanning (this bit=1 implies that the sensor accepts the 'enable/disable scanning' bit in the <i>Set Sensor Event Enable</i> command). [5] - Init Events 1b = enable events (per Sensor Event Message Control Support bits in Sensor Capabilities field, and per the Event Mask fields, below). [4] - Init Thresholds 1b = initialize sensor thresholds (per settable threshold mask below). [3] - Init Hysteresis 1b = initialize sensor hysteresis (per Sensor Hysteresis Support bits in the Sensor Capabilities field, below). [2] - Init Sensor Type 1b = initialize Sensor Type and Event / Reading Type code <u>Sensor Default (power up) State</u> Reports how this sensor comes up on device power up and hardware/cold reset. The Initialization Agent does not use this bit. This bit solely reports to software how the sensor comes prior to being initialized by the Initialization Agent. [1] - 0b = event generation disabled, 1b = event generation enabled [0] - 0b = sensor scanning disabled, 1b = sensor scanning enabled

byte	Field Name	size	Description
12	Sensor Capabilities	1	<p>[7] - 1b = Ignore sensor if Entity is not present or disabled. 0b = don't ignore sensor</p> <p><b>Sensor Auto Re-arm Support</b> Indicates whether the sensor requires manual rearming, or automatically rearms itself when the event clears. 'manual' implies that the <i>get sensor event status</i> and <i>rearm sensor events</i> commands are supported [6] - 0b = no (manual), 1b = yes (auto)</p> <p><b>Sensor Hysteresis Support</b> [5:4] - 00b = No hysteresis, or hysteresis built-in but not specified. 01b = hysteresis is readable. 10b = hysteresis is readable and settable. 11b = Fixed, unreadable, hysteresis. Hysteresis fields values implemented in the sensor.</p> <p><b>Sensor Threshold Access Support</b> [3:2] - 00b = no thresholds. 01b = thresholds are readable, per Reading Mask, below. 10b = thresholds are readable and settable per Reading Mask and Settable Threshold Mask, respectively. 11b = Fixed, unreadable, thresholds. Which thresholds are supported is reflected by the Reading Mask. The threshold value fields report the values that are 'hard-coded' in the sensor.</p> <p><b>Sensor Event Message Control Support</b> Indicates whether this sensor generates Event Messages, and if so, what type of Event Message control is offered. [1:0] - 00b = per threshold/discrete-state event enable/disable control (implies that entire sensor and global disable are also supported) 01b = entire sensor only (implies that global disable is also supported) 10b = global disable only 11b = no events from sensor</p>
13	Sensor Type	1	Code representing the sensor type. From Table 36-3, Sensor Type Codes. E.g. Temperature, Voltage, Processor, etc.
14	Event / Reading Type Code	1	Event/Reading Type Code. From Table 36-1, Event/Reading Type Code Ranges.

byte	Field Name	size	Description
15 16	Assertion Event Mask / Lower Threshold Reading Mask	2	<p>This field reports the assertion event generation or threshold event generation capabilities for a discrete or threshold-based sensor, respectively. This field is also used by the init agent to enable assertion event generation when the 'Init Events' bit in the Sensor Capabilities field is set and the Sensor Event Message Control Support field indicates that the sensor has 'per threshold/discrete state' event enable control.</p> <p><u>Assertion Event Mask (for non- threshold-based sensors)</u> The Event Mask bytes are a bit mask that specifies support for 15 successive events starting with the event specified by Event/Reading Type Code. LS byte first. [15] - reserved. Write as '0'. [14:0] - Event offsets 14 through 0, respectively. 1b = assertion event can be generated by this sensor</p> <p><u>Lower Threshold Reading Mask (for threshold-based sensors)</u> Indicates which lower threshold comparison status is returned via the <i>Get Sensor Reading</i> command. [15] - reserved. Write as 0b [14] - 1b = Lower non-recoverable threshold comparison is returned [13] - 1b = Lower critical threshold is comparison returned [12] - 1b = Lower non-critical threshold is comparison returned</p> <p><u>Threshold Assertion Event Mask (for threshold-based sensors)</u> [11] - 1b = assertion event for upper non-recoverable going high supported [10] - 1b = assertion event for upper non-recoverable going low supported [9] - 1b = assertion event for upper critical going high supported [8] - 1b = assertion event for upper critical going low supported [7] - 1b = assertion event for upper non-critical going high supported [6] - 1b = assertion event for upper non-critical going low supported [5] - 1b = assertion event for lower non-recoverable going high supported [4] - 1b = assertion event for lower non-recoverable going low supported [3] - 1b = assertion event for lower critical going high supported [2] - 1b = assertion event for lower critical going low supported [1] - 1b = assertion event for lower non-critical going high supported [0] - 1b = assertion event for lower non-critical going low supported</p>
17	Deassertion Event Mask / Upper		<u>Deassertion Event Mask (for non- threshold-based sensors)</u>

18	Threshold Reading Mask	1	<p>The Event Mask bytes are a bit mask that specifies support for 15 successive events starting with the event specified by Event/Reading Type Code. LS byte first.</p> <p>[15] - reserved. Write as 0b</p> <p>[14:0] - Event offsets 14 through 0, respectively.</p> <p>1b = assertion event can be generated for this state.</p> <p><u>Upper Threshold Reading Mask (for threshold-based sensors)</u></p> <p>Indicates which upper threshold comparison status is returned via the <i>Get Sensor Reading</i> command.</p> <p>[15] - reserved. Write as 0b</p> <p>[14] - 1b = Upper non-recoverable threshold comparison is returned</p> <p>[13] - 1b = Upper critical threshold is comparison returned</p> <p>[12] - 1b = Upper non-critical threshold is comparison returned</p> <p><b>Threshold Deassertion Event Mask</b></p> <p>[11] - 1b = deassertion event for upper non-recoverable going high supported</p> <p>[10] - 1b = deassertion event for upper non-recoverable going low supported</p> <p>[9] - 1b = deassertion event for upper critical going high supported</p> <p>[8] - 1b = deassertion event for upper critical going low supported</p> <p>[7] - 1b = deassertion event for upper non-critical going high supported</p> <p>[6] - 1b = deassertion event for upper non-critical going low supported</p> <p>[5] - 1b = deassertion event for lower non-recoverable going high supported</p> <p>[4] - 1b = deassertion event for lower non-recoverable going low supported</p> <p>[3] - 1b = deassertion event for lower critical going high supported</p> <p>[2] - 1b = deassertion event for lower critical going low supported</p> <p>[1] - 1b = deassertion event for lower non-critical going high supported</p> <p>[0] - 1b = deassertion event for lower non-critical going low supported</p>
19	Discrete Reading Mask /		<u>Reading Mask (for non- threshold based sensors)</u>

20	SettableThreshold Mask, ReadableThreshold Mask	2	<p>Indicates what discrete readings can be returned by this sensor, or, for thresholdbased sensors, this indicates which thresholds are settable and which are readable. The Reading Mask bytes are a bit mask that specifies support for 15 successive states starting with the value from <i>Table 36-1, Event/Reading Type Code Ranges</i>.LS bytefirst.</p> <p>[15] - reserved. Write as 0b [14:0] - state bits 0 through 14. 1b = discrete state can be returned by this sensor.</p> <p><u>Settable Threshold Mask (for threshold-based sensors)</u> Indicates which thresholds are settable via the <i>Set Sensor Thresholds</i>.This mask also indicates which threshold values will be initialized if the 'Init Events' bit is set. LS byte first. [15:14] - reserved. Write as 00b. [13] - 1b = Upper non-recoverable threshold is settable [12] - 1b = Upper critical threshold is settable [11] - 1b = Upper non-critical threshold is settable [10] - 1b = Lower non-recoverable threshold is settable [9] - 1b = Lower critical threshold is settable [8] - 1b = Lower non-critical threshold is settable</p> <p><u>Readable Threshold Mask (for threshold-based sensors)</u> Indicates which thresholds are readable via the <i>Get Sensor Thresholds</i> command. [7:6] - reserved. Write as 00b. [5] - 1b = Upper non-recoverable threshold is readable [4] - 1b = Upper critical threshold is readable [3] - 1b = Upper non-critical threshold is readable [2] - 1b = Lower non-recoverable threshold is readable [1] - 1b = Lower critical threshold is readable [0] - 1b = Lower non-critical threshold is readable</p>
21	Sensor Units 1	1	<p>[7:6] - <u>Analog (numeric) Data Format**</u> 00b = unsigned 01b = 1's complement (signed) 10b = 2's complement (signed) 11b = Does not return analog (numeric) reading</p> <p>[5:3] - <u>Rate unit</u> 000b = none 001b = per <math>\mu</math>S 010b = per ms 011b = per s 100b = per minute 101b = per hour 110b = per day 111b = reserved</p> <p>[2:1] - <u>Modifier unit</u> 00b = none 01b = Basic Unit / Modifier Unit 10b = Basic Unit * Modifier Unit 11b = reserved</p> <p>[0] - <u>Percentage</u> 0b = no, 1b = yes</p> <p>** Specifies threshold and 'analog' reading, if 'analog' reading provided. If neither thresholds nor analog reading are provided, this field should be written as 00h.</p>
22	Sensor Units 2 - Base Unit	1	[7:0] - Units Type code: See Table 37-14, Sensor Unit Type Codes.
23	Sensor Units 3 - Modifier Unit	1	[7:0] - Units Type code, 00h if unused.
24	Linearization	1	[7] - reserved [6:0] - enum (linear, ln, log10, log2, e, exp10, exp2, 1/x, sqrt(x), cube(x), sqrt(x), cube -1 (x) ) - 70h = non-linear. 71h-7Fh = non-linear, OEM defined.
25	M	1	[7:0] - M: LS 8 bits [2's complement, signed, 10 bit 'M' value.] -26
26	M, Tolerance	1	[7:6] - M: MS 2 bits [5:0] - Tolerance: 6 bits, unsigned (Tolerance in +/- 1/2 raw counts)
27	B	1	[7:0] - B: LS 8 bits [2's complement, signed, 10-bit 'B' value.]
28	B, Accuracy	1	[7:6] - B: MS 2 bits Unsigned, 10-bit Basic Sensor Accuracy in 1/100 percent scaled up by unsigned

			Accuracy exponent: [5:0] - Accuracy: LS 6 bits
29	Accuracy, Accuracy exp	1	[7:4] - Accuracy: MS 4 bits [3:2] - Accuracy exp: 2 bits, unsigned [1:0] - reserved: 2 bits
30	R exp, B exp	1	[7:4] - R (result) exponent 4 bits, signed [3:0] - B exponent 4 bits, signed
31	Analog characteristic flags	1	[7:3] - reserved [2] - normal min specified 1b = yes, 0b = normal min field unspecified [1] - normal max specified 1b = yes, 0b = normal max field unspecified [0] - nominal reading specified 1b = yes, 0b = nominal reading field unspecified
32	Nominal Reading	1	Given as a raw value. Must be converted to units-based value using the 'y=Mx+B' formula. 1's or 2's complement signed or unsigned per flag bits in Sensor Units 1.
33	Normal Maximum	1	Given as a raw value. Must be converted to units-based value using the 'y=Mx+B' formula. 1's or 2's complement signed or unsigned per 'signed' bit in Sensor Units 1.
34	Normal Minimum	1	Given as a raw value. Must be converted to units-based value using the 'y=Mx+B' formula. Signed or unsigned per 'signed' bit in Sensor Units 1.
35	Sensor Maximum Reading	1	Given as a raw value. Must be converted to units-based value based using the y=Mx+B formula. Signed or unsigned per 'signed' bit in sensor flags. Normally 'FFh' for an 8-bit unsigned sensor, but can be a lesser value if the sensor has a restricted range. If max. reading cannot be pre-specified this value should be set to max value, based on data format, (e.g. FFh for an unsigned sensor, 7Fh for 2's complement, etc.)
36	Sensor Minimum Reading	1	Given as a raw value. Must be converted to units-based value using the 'y=Mx+B' formula. Signed or unsigned per 'signed' bit in sensor flags. If min. reading cannot be pre-specified this value should be set to min value, based on data format, (e.g. 00h for an unsigned sensor, 80h for 2's complement, etc.)
37	Upper non-recoverable Threshold	1	Use of this field is based on Settable Threshold Mask. If the corresponding bit is set in the mask byte and the 'Init Sensor Thresholds' bit is also set, then this value will be used for initializing the sensor threshold. Otherwise, this value should be ignored. The thresholds are given as raw values that must be converted to units-based values using the 'y=Mx+B' formula.
38	Upper critical Threshold	1	Use of this field is based on Settable Threshold Mask, above
39	Upper non-critical Threshold	1	Use of this field is based on Settable Threshold Mask, above
40	Lower non-recoverable Threshold	1	Use of this field is based on Settable Threshold Mask, above
41	Lower critical Threshold	1	Use of this field is based on Settable Threshold Mask, above
42	Lower non-critical Threshold	1	Use of this field is based on Settable Threshold Mask, above
43	Positive-going Threshold Hysteresis value	1	Positive hysteresis is defined as the unsigned number of counts that are subtracted from the raw threshold values to create the 're-arm' point for all positive-going thresholds on the sensor. 0 indicates that there is no hysteresis on positive-going thresholds for this sensor. Hysteresis values are given as raw counts. That is, to find the degree of hysteresis in units, the value must be converted using the 'y=Mx+B' formula.
44	Negative-going Threshold Hysteresis value	1	Negative hysteresis is defined as the unsigned number of counts that are added to the raw threshold value to create the 're-arm' point for all negative-going thresholds on the sensor. 0 indicates that there is no hysteresis on negative-going thresholds for this sensor.
45	reserved	1	reserved. Write as 00h.
46	reserved	1	reserved. Write as 00h.
47	OEM	1	Reserved for OEM use.
48	ID String Type/Length Code	1	Sensor 'ID' String Type/Length Code, per Section 37.14, Type/Length Byte Format.
49: +N	ID String Bytes	N	Sensor ID String bytes. Only present if non-zero length in Type/Length field. 16 bytes, maximum. Note: the SDR can be implemented as a fixed length record. Bytes beyond the ID string

			<i>bytes are unspecified and should be ignored.</i>
--	--	--	---

## Sensor Data Record Type 12h IPMB management Controller Device Locator Record

byte	Field Name	Size	Description
<b>RECORD HEADER</b>			
1	Record ID – L	2	The Record ID is used by the Sensor Data Repository device for record organization and access. It is <i>not</i> related to the sensor ID.
2	Record ID – H		
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. ( <b>51h</b> for this specification) <i>BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.</i>
4	Record Type	1	Record Type Number = 12h, Management Controller Locator
5	Record Length	1	Number of remaining record bytes following.
<b>RECORD KEY BYTES</b>			
6	Device Slave Address	1	[7:1] - 7-bit I <sup>2</sup> C Slave Address [1] of device on channel. [0] - Reserved
7	Channel Number	1	[7:4] - reserved [3:0] - Channel number for the channel that the management controller is on. Use 0h for the primary BMC. (New byte for IPMI v1.5. Note this addition causes some of the following byte offsets to be pushed down when compared to the IPMI v1.0 version of this record.)
<b>RECORD BODY BYTES</b>			
8	Power State Notification Global Initialization	1	<p>Power State Notification</p> <p>[7] - 1b = ACPI System Power State notification required (by system s/w) 0b = no ACPI System Power State notification required</p> <p>[6] - 1b = ACPI Device Power State notification required (by system s/w) 0b = no ACPI Device Power State notification required</p> <p>[5] - For backward compatibility, this bit does not apply to the BMC, and should be written as 0b. 0b = Dynamic controller - controller may or may not be present. Software should not generate error status if this controller is not present. 1b = Static controller - this controller is expected to be present in the system at all times. Software may generate an error status if controller is not detected.</p> <p>[4] – reserved</p> <p>Global Initialization</p> <p>[3] - 1b = Controller logs Initialization Agent errors (only applicable to Management Controller that implements the initialization agent function. Set to 0b otherwise.)</p> <p>[2] - 1b = Log Initialization Agent errors accessing this controller (this directs the initialization agent to log any failures setting the Event Receiver)</p> <p>[1:0] - 00b = Enable event message generation from controller (Init agent will set Event Receiver address into controller) 01b = Disable event message generation from controller (Init agent will set Event Receiver to FFh). This provides a temporary fix for a broken controller that floods the system with events. It can also be used for development / debug purposes. 10b = Do not initialize controller. This selection is for development / debug support. 11b = reserved.</p>

byte	Field Name	size	Description
9	Device Capabilities	1	Device Support [7] - 1b = Chassis Device. (device functions as chassis device, per ICMB spec) [6] - 1b = Bridge [5] - 1b = IPMB Event Generator (device generates event messages on IPMB) [4] - 1b = IPMB Event Receiver (device accepts event messages from IPMB) [3] - 1b = FRU Inventory Device (accepts FRU commands to FRU Device #0 at LUN 00b) [2] - 1b = SEL Device (provides interface to SEL) [1] - 1b = SDR Repository Device (For BMC, indicates BMC provides interface to 1b = SDR Repository. For other controller, indicates controller accepts Device SDR commands) [0] - 1b = Sensor Device (device accepts sensor commands) See Table 37-11, IPMB/I <sub>2</sub> C Device Type Codes
10	reserved	1	reserved
11	reserved	1	reserved
12	reserved	1	reserved
13	Entity ID	1	Entity ID for the FRU associated with this device. 00h if not specified. If device supports FRU commands at LUN 00b, this Entity ID applies to both the IPM device and the FRU information accessed via LUN 00b.
14	Entity Instance	1	Instance number for entity.
15	OEM	1	Reserved for OEM use.
16	Device ID String Type/Length	1	Device ID String Type/Length code per Section 37.14, Type/Length Byte Format.
17:+N	Device ID String	1	Short 'ID' string for the device. 16 bytes, maximum.

Notes:

1. 7-bit I<sub>2</sub>C Slave Address field. By convention, the I<sub>2</sub>C slave address is represented as an eight-bit number with the least-significant bit always 0. E.g. 20h = 00100000b. The 7-bit Slave Address field holds the most-significant 7 bits of this value. E.g. 0010000b.

## Appendix C

**Table C-1, Event/Reading Type Code Ranges**  
(Table 36-1 in IPMI V1.5)

Event/Reading Type Code category	7-bit Event/Reading Type Code	Sensor Class	Description
unspecified	00h	n/a	Event/Reading Type unspecified.
Threshold	01h	Threshold	Indicates a sensor that threshold states in sensor access and/or events.
Generic	02h-0Bh	discrete	Event/Reading Type code & State bit positions / event offsets specified in the Table B-2, Generic Event/Reading Type Codes, below.
Sensor-specific	6Fh	discrete	Indicates that the discrete state information is specific to the sensor type. State bit positions / event offsets for a particular sensor type are specified in the 'sensor-specific offset' column in Table B-3, Sensor Type Codes, below.
OEM	70h-7Fh	OEM	Indicates that the discrete state information is specific to the OEM identified by the Manufacturer ID for the IPM device that is providing access to the sensor.

**Table C-2, Generic Event/Reading Type Codes**  
(Table 36-2 in IPMI V1.5)

Generic Event/Reading Type Code	Event/Reading Class	Generic Offset	Description
THRESHOLD BASED STATES			
01h	Threshold	00h	Lower Non-critical - going low
		01h	Lower Non-critical - going high
		02h	Lower Critical - going low
		03h	Lower Critical - going high
		04h	Lower Non-recoverable - going low
		05h	Lower Non-recoverable - going high
		06h	Upper Non-critical - going low
		07h	Upper Non-critical - going high
		08h	Upper Critical - going low
		09h	Upper Critical - going high
		0Ah	Upper Non-recoverable - going low
0Bh	Upper Non-recoverable - going high		
DMI-based "Usage State" STATES			
02h	Discrete	00h	Transition to Idle
		01h	Transition to Active
		02h	Transition to Busy
DIGITAL/DISCRETE EVENT STATES			
03h	'digital' Discrete	00h	State Deasserted
		01h	State Asserted
04h	'digital' Discrete	00h	Predictive Failure deasserted
		01h	Predictive Failure asserted
05h	'digital' Discrete	00h	Limit Not Exceeded
		01h	Limit Exceeded
06h	'digital' Discrete	00h	Performance Met
		01h	Performance Lags
DMI-based SEVERITY EVENT STATES			
07h	Discrete	00h	transition to OK
		01h	transition to Non-Critical from OK
		02h	transition to Critical from less severe
		03h	transition to Non-recoverable from less severe
		04h	transition to Non-Critical from more severe
		05h	transition to Critical from Non-recoverable
		06h	transition to Non-recoverable
		07h	Monitor
		08h	Informational
DMI-based AVAILABILITY STATUS STATES			
08h	'digital' Discrete	00h	Device Removed / Device Absent
		01h	Device Inserted / Device Present
09h	'digital' Discrete	00h	Device Disabled
		01h	Device Enabled
0Ah	Discrete	00h	transition to Running
		01h	transition to In Test
		02h	transition to Power Off
		03h	transition to On Line
		04h	transition to Off Line
		05h	transition to Off Duty
		06h	transition to Degraded
		07h	transition to Power Save
		08h	Install Error
Other AVAILABILITY STATUS STATES			

0Bh	Discrete	00h 01h 02h	Redundancy Regained Redundancy Lost Redundancy Degraded
-----	----------	-------------------	---

**Table C-3, Sensor Type Codes**  
(Table 36-3 in IPMI V1.5)

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
Reserved	00h	-	reserved
Temperature	01h	-	Temperature
Voltage	02h	-	Voltage
Current	03h	-	Current
Fan	04h	-	Fan
Physical Security (Chassis Intrusion)	05h	00h 01h 02h 03h 04h 05h 06h	General Chassis Intrusion Drive Bay intrusion I/O Card area intrusion Processor area intrusion LAN Leash Lost (system has been unplugged from LAN) Unauthorized dock/undock FAN area intrusion (supports detection of hot plug fan tampering)
Platform Security Violation Attempt	06h	00h 01h 02h 03h 04h 05h	Secure Mode (Front Panel Lockout) Violation attempt Pre-boot Password Violation - user password Pre-boot Password Violation attempt - setup password Pre-boot Password Violation - network boot password Other pre-boot Password Violation Out-of-band Access Password Violation
Processor	07h	00h 01h 02h 03h  04h 05h 06h 07h 08h 09h	IERR Thermal Trip FRB1/BIST failure FRB2/Hang in POST failure (used hang is believed to be due or related to a processor failure. Use System Firmware Progress sensor for other BIOS hangs.) FRB3/Processor Startup/Initialization failure (CPU didn't start) Configuration Error (for DMI) SM BIOS 'Uncorrectable CPU-complex Error' Processor Presence detected Processor disabled Terminator Presence Detected
Power Supply	08h	00h 01h 02h 03h 04h 05h	Presence detected Power Supply Failure detected Predictive Failure asserted Power Supply AC lost AC lost or out-of-range AC out-of-range, but present
Power Unit	09h	00h 01h 02h 03h 04h 05h 06h	Power Off / Power Down Power Cycle 240VA Power Down Interlock Power Down AC lost Soft Power Control Failure (unit did not respond to request to turn on) Power Unit Failure detected
Cooling Device	0Ah	-	-
Other Units-based Sensor (per units given in SDR)	0Bh	-	-
Memory	0Ch	00h 01h 02h 03h 04h 05h	Correctable ECC / other correctable memory error Uncorrectable ECC / other uncorrectable memory error Parity Memory Scrub Failed (stuck bit) Memory Device Disabled Correctable ECC / other correctable memory error logging limit reached <i>The Event Data 3 field for this command can be used to provide an event extension code, with the following definition: 7:0 DIMM/SIMM/RIMM identification, relative to the entity that the sensor is associated with (if SDR provided for this sensor)</i>

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
Drive Slot (Bay)	0Dh	-	-
POST Memory Resize	0Eh	-	-
System Firmware Progress (formerly POST Error)	0Fh	00h	System Firmware Error (POST Error) <i>The Event Data 2 field can be used to provide an event extension code, with the following definition:</i> Event Data 2: 00h Unspecified. 01h No system memory is physically installed in the system. 02h No usable system memory, all installed memory has experienced an unrecoverable failure. 03h Unrecoverable hard-disk/ATAPI/IDE device failure. 04h Unrecoverable system-board failure. 05h Unrecoverable diskette subsystem failure. 06h Unrecoverable hard-disk controller failure. 07h Unrecoverable PS/2 or USB keyboard failure. 08h Removable boot media not found 09h Unrecoverable video controller failure 0Ah No video device detected. 0Bh to FFh reserved
		01h	System Firmware Hang (uses same Event Data 2 definition as following System Firmware Progress offset)
		02h	System Firmware Progress <i>The Event Data 2 field can be used to provide an event extension code, with the following definition:</i> Event Data 2: 00h Unspecified. 01h Memory initialization. 02h Hard-disk initialization 03h Secondary processor(s) initialization 04h User authentication 05h User-initiated system setup 06h USB resource configuration 07h PCI resource configuration 08h Option ROM initialization 09h Video initialization 0Ah Cache initialization 0Bh SM Bus initialization 0Ch Keyboard controller initialization 0Dh Embedded controller/management controller Initialization 0Eh Docking station attachment 0Fh Enabling docking station 10h Docking station ejection 11h Disabling docking station 12h Calling operating system wake-up vector 13h Starting operating system boot process, e.g. calling Int 19h 14h to FFh reserved.
Event Logging Disabled	10h	00h 01h	Correctable Memory Error Logging Disabled Event 'Type' Logging Disabled. Event Logging is disabled for the following event/reading type and offset has been disabled. <b>Event Data 2</b> Event/Reading Type Code <b>Event Data 3</b> [7:6] - reserved. Write as 00b. [5] 1b = logging has been disabled for all events of given type [4] 1b = assertion event, 0b = de-assertion event [3:0] Event Offset Log Area Reset/Cleared All Event Logging Disabled
		02h 03h	

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
Watchdog 1	11h	00h 01h 02h 03h 04h 05h 06h 07h	This sensor is provided to support IPMI v0.9 to v1.0 transition. This is deprecated in IPMI v1.5. See sensor 23h for recommended definition of Watchdog sensor for new v1.0 and for IPMI v1.5 implementations.  BIOS Watchdog Reset OS Watchdog Reset OS Watchdog Shut Down OS Watchdog Power Down OS Watchdog Power Cycle OS Watchdog NMI / Diagnostic Interrupt OS Watchdog Expired, status only OS Watchdog pre-timeout Interrupt, non-NMI
System Event	12h	00h 01h 02h	System Reconfigured OEM System Boot Event Undetermined system hardware failure (this event would typically require system-specific diagnostics to determine FRU / failure type)
Critical Interrupt	13h	00h 01h 02h 03h 04h 05h 06h 07h 08h 09h	Front Panel NMI / Diagnostic Interrupt Bus Timeout I/O channel check NMI Software NMI PCI PERR PCI SERR EISA Fail Safe Timeout Bus Correctable Error Bus Uncorrectable Error Fatal NMI (port 61h, bit 7)
Button	14h	00h 01h 02h	Power Button pressed Sleep Button pressed Reset Button pressed
Module / Board	15h	-	-
Microcontroller/ Coprocessor	16h	-	-
Add-in Card	17h	-	-
Chassis	18h	-	-
Chip Set	19h	-	-
Other FRU	1Ah	-	-
Cable / Interconnect	1Bh	-	-
Terminator	1Ch	-	-
System Boot Initiated	1Dh	00h 01h 02h 03h 04h	Initiated by power up Initiated by hard reset Initiated by warm reset User requested PXE boot Automatic boot to diagnostic
Boot Error	1Eh	00h 01h 02h 03h 04h	No bootable media Non-bootable diskette left in drive PXE Server not found Invalid boot sector Timeout waiting for user selection of boot source
OS Boot	1Fh	00h 01h 02h 03h 04h 05h 06h	A: boot completed C: boot completed PXE boot completed Diagnostic boot completed CD-ROM boot completed ROM boot completed boot completed - boot device not specified
OS Critical Stop	20h	00h 01h	Stop during OS load / initialization Run-time Stop

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
Slot / Connector	21h	00h 01h 02h 03h 04h 05h 06h 07h 08h	Fault Status asserted Identify Status asserted Slot / Connector Device installed/attached [This can include dock events] Slot / Connector Ready for Device Installation - Typically, this means that the slot power is off. The Ready for Installation, Ready for Removal, and Slot Power states can transition together, depending on the slot implementation. Slot/Connector Ready for Device Removal Slot Power is Off Slot / Connector Device Removal Request - This is typically connected to a switch that becomes asserted to request removal of the device) Interlock asserted - This is typically connected to a switch that mechanically enables/disables power to the slot, or locks the slot in the 'Ready for Installation / Ready for Removal states' – depending on the slot implementation. The asserted state indicates that the lock-out is active. Slot is Disabled <i>The Event Data 2 &amp; 3 fields can be used to provide an event extension code, with the following definition:</i> <div style="text-align: center;"><b>Event Data 2</b></div> 7 reserved 6:0 Slot/Connector Type 0 PCI 1 Drive Array 2 External Peripheral Connector 3 Docking 4 other standard internal expansion slot 5 slot associated with entity specified by Entity ID for sensor all other = reserved <div style="text-align: center;"><b>Event Data 3</b></div> 7:0 Slot/Connector Number
System ACPI Power State	22h	00h 01h 02h 03h 04h 05h 06h 07h 08h 09h 0Ah 0Bh 0Ch 0Eh	S0 / G0 "working" S1 "sleeping with system h/w & processor context maintained" S2 "sleeping, processor context lost" S3 "sleeping, processor & h/w context lost, memory retained." S4 "non-volatile sleep / suspend-to disk" S5 / G2 "soft-off" S4 / S5 soft-off, particular S4 / S5 state cannot be determined G3 / Mechanical Off Sleeping in an S1, S2, or S3 states (used when particular S1, S2, S3 state cannot be determined) G1 sleeping (S1-S4 state cannot be determined) S5 entered by override Legacy ON state Legacy OFF state Unknown

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
Watchdog 2	23h	00h 01h 02h 03h 04h-07h 08h	<p>This sensor is recommended for new IPMI v1.0 and later implementations.</p> <p>Timer expired, status only (no action, no interrupt)</p> <p>Hard Reset</p> <p>Power Down</p> <p>Power Cycle</p> <p>reserved</p> <p>Timer interrupt</p> <p>The Event Data 2 field for this command can be used to provide an event extension code, with the following definition:</p> <p>7:4 interrupt type</p> <p>0h = none</p> <p>1h = SMI</p> <p>2h = NMI</p> <p>3h = Interrupt</p> <p>Fh = unspecified</p> <p>all other = reserved</p> <p>3:0 timer use at expiration:</p> <p>0h = reserved</p> <p>1h = BIOS FRB2</p> <p>2h = BIOS/POST</p> <p>3h = OS Load</p> <p>4h = SMS/OS</p> <p>5h = OEM</p> <p>Fh = unspecified</p> <p>all other = reserved</p>
Platform Alert	24h	00h 01h 02h 03h	<p>This sensor can be used for returning the state and generating events associated with alerts that have been generated by the platform mgmt. subsystem</p> <p>platform generated page</p> <p>platform generated LAN alert</p> <p>Platform Event Trap generated, formatted per IPMI PET specification</p> <p>platform generated SNMP trap, OEM format</p>
Entity Presence	25h	00h 01h	<p>This sensor type provides a mechanism that allows a management controller to direct system management software to ignore a set of sensors based on detecting that presence of an entity. This sensor type is not typically used for event generation - but to just provide a present reading.</p> <p>Entity Present. This indicates that the Entity identified by the Entity ID for the sensor is present.</p> <p>Entity Absent. This indicates that the Entity identified by the Entity ID for the sensor is absent. If the entity is absent, system management software should consider all sensors associated with that Entity to be absent as well – and ignore those sensors.</p>
Monitor ASIC / IC	26h	-	-
LAN	27h	00h 01h	<p>LAN Heartbeat Lost</p> <p>LAN Heartbeat</p>
Management Subsystem Health	28h	00h 01h 02h 03h	<p>sensor access degraded or unavailable</p> <p>controller access degraded or unavailable</p> <p>management controller off-line</p> <p>management controller unavailable</p>
Reserved	remaining	-	
OEM RESERVED	C0h-FFh	-	

## Appendix D – Entity ID's

Table 37-12, IPMI V1.5)

Code		Entity
0*	00h	Unspecified
1*	01h	Other
2*	02h	unknown (unspecified)
3*	03h	Processor
4*	04h	disk or disk bay
5*	05h	peripheral bay
6*	06h	system management module
7*	07h	system board (main system board, may also be a processor board and/or internal expansion board)
8*	08h	memory module (board holding memory devices)
9*	09h	processor module (holds processors, use this designation when processors are not mounted on system board)
10*	0Ah	power supply (DMI refers to this as a "power unit", but it's used to represent a power supply). Use this value for the main power supply (supplies) for the system.
11	0Bh	add-in card
12	0Ch	front panel board (control panel)
13	0Dh	back panel board
14	0Eh	power system board
15	0Fh	drive backplane
16	10h	system internal expansion board (contains expansion slots).
17	11h	Other system board (part of board set)
18	12h	processor board (holds 1 or more processors - includes boards that hold SECC modules)
19	13h	power unit / power domain (typically used as a pre-defined logical entity for grouping power supplies)
20	14h	power module / DC-to-DC converter. Use this value for internal converters. Note: You should use Entity ID 10 (power supply) for the main power supply even if the main supply is a DC-to-DC converter, e.g. gets external power from a -48 DC source.
21	15h	power management / power distribution board
22	16h	chassis back panel board
23	17h	system chassis
24	18h	sub-chassis
25	19h	Other chassis board
26	1Ah	Disk Drive Bay
27	1Bh	Peripheral Bay
28	1Ch	<i>Device Bay</i>
29	1Dh	fan / cooling device
30	1Eh	cooling unit (can be used as a pre-defined logical entity for grouping fans or other cooling devices)
31	1Fh	cable / interconnect
32	20h	memory device (This Entity ID should be used for replaceable memory devices, e.g. DIMM/SIMM. It is recommended that Entity IDs not be used for individual non-replaceable memory devices. Rather, monitoring and error reporting should be associated with the FRU [e.g. memory card] holding thememory.)
33	21h	System Management Software
34	22n	BIOS
35	23h	Operating System
36	24h	system bus
37	25h	Group - this is a logical entity for use with Entity Association records. It is provided to allow a sensor data record to point to an Entity-association record when there is no appropriate pre-defined logical entity for the entity grouping. This Entity should not be used as a physical entity.
38	26h	Remote (Out of Band) Management Communication
	90h - AFh	Chassis-specific Entities. These IDs are system specific and can be assigned by the chassis provider.
	B0h - CFh	Board-set specific Entities. These IDs are system specific and can be assigned by the Board-set provider.
	D0h - FFh	OEM System Integrator defined. These IDs are system specific and can be assigned by the system\ integrator, or OEM.
-		all other values reserved

**Appendix E – Sensor Unit Type Codes**  
(Table 37-14 IPMI V1.5)

Code	Unit	Code	Unit	Code	Unit
0	unspecified	34	m	68	megabit
1	degrees C	35	cu cm	69	gigabit
2	degrees F	36	cu m	70	byte
3	degrees K	37	liters	71	kilobyte
4	Volts	38	fluid ounce	72	megabyte
5	Amps	39	radians	73	gigabyte (data)
6	Watts	40	steradians	74	word (data)
7	Joules	41	revolutions	75	dword
8	Coulombs	42	cycles	76	qword
9	VA	43	gravities	77	line (re. mem. line)
10	Nits	44	ounce	78	hit
11	lumen	45	pound	79	miss
12	lux	46	ft-lb	80	retry
13	Candela	47	oz-in	81	reset
14	kPa	48	gauss	82	overflow / overrun
15	PSI	49	gilberts	83	underrun
16	Newton	50	henry	84	collision
17	CFM	51	millihenry	85	packets
18	RPM	52	farad	86	messages
19	Hz	53	microfarad	87	characters
20	microsecond	54	ohms	88	error
21	millisecond	55	siemens	89	correctable error
22	second	56	mole	90	uncorrectable error
23	minute	57	becquerel	91	
24	hour	58	PPM (parts/million)	92	
25	day	59	reserved	93	
26	week	60	Decibels	94	
27	mil	61	DbA	95	
28	inches	62	DbC	96	
29	feet	63	gray	97	
30	cu in	64	sievert	98	
31	cu feet	65	color temp deg K	99	
32	mm	66	bit	100	
33	cm	67	kilobit	101	

## Appendix F - Sensor Reading Conversion Formula

The following presents the formula used for converting 'raw' sensor readings for linear and linearized sensors to real values in the desired 'units' for the sensor (e.g. Volts, Amps, etc.).

$$y = L[(Mx + (B * 10^{K1})) * 10^{K2}] \text{ units}$$

where:

**x** Raw reading

**y** Converted reading

**L [ ]** Linearization function specified by 'linearization type'.  
This function is 'null' ( $y = f(x) = x$ ) if the sensor is linear.

**M** Signed integer constant multiplier

**B** Signed additive 'offset'

**K1** Signed Exponent. Sets 'decimal point' location for B. This is called the 'B' exponent in the SDRs.

**K2** Signed *Result* Exponent. Sets 'decimal point' location for the result before the linearization function is applied. This is called the 'R' exponent in SDRs. Linear and Linearized readings have constant accuracy, tolerance, M, and B factors regardless of the reading. Accuracy, tolerance, M, and B for 'Non-linear' sensors are only valid at the nominal reading.

Note: The analog sensors in this application are all linear without an offset. As a result, the formula reduces to:

$$y = (Mx) * 10^{K2} \text{ units}$$

## Appendix G FRU Storage information

### G-1. Common Header format:

field length	field
1*	Common Header Format Version 7:4 - reserved, write as 0000b 3:0 - format version number = 1h for this specification.
1*	Internal Use Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.
1*	Chassis Info Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.
1*	Board Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.
1*	Product Info Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.
1*	MultiRecord Area Starting Offset (in multiples of 8 bytes). 00h indicates that this area is not present.
1*	PAD, write as 00h
1*	Common Header Checksum (zero checksum)

\* = These fields are MANDATORY

### G-2. TYPE/LENGTH BYTE FORMAT

The following presents the specification of the type/length byte.

#### 7:6 - type code

- 00 - binary or unspecified
- 01 - BCD plus (see below)
- 10 - 6-bit ASCII, packed (overrides Language Codes)
- 11 - Interpretation depends on Language Codes. 11b indicates 8-bit ASCII + Latin 1 if the Language Code is English for the area or record containing the field, or 2-byte UNICODE (least significant byte first) if the Language Code is not English. At least two bytes of data must be present when this type is used. Therefore, the length (number of data bytes) will always be >1 if data is present, 0 if data is not present.

#### 5:0 - number of data bytes.

- 000000 indicates that the field is empty. When the type code is 11b, a length of 000001 indicates 'end of fields'. I.e. Type/Length = C1h indicates 'end of fields'.

"ASCII+LATIN1" is derived from the first 256 characters of Unicode 2.0. The first 256 codes of Unicode follow ISO 646 (ASCII) and ISO 8859/1 (Latin 1). The Unicode "C0 Controls and Basic Latin" set defines the first 128 8-bit characters (00h-7Fh) and the "C1 Controls and Latin-1 Supplement" defines the second 128 (80h-FFh).

"6-bit ASCII" is the 64 characters starting from character 20h (space) from the ASCII+LATIN1 set. So 6-bit ASCII value 000000b maps to 20h (space), 000001b maps to 21h (!), etc. Packed 6-bit

ASCII takes the 6-bit characters and packs them 4 characters to every 3 bytes, with the first character in the least significant 6-bits of the first byte. A table of 6-bit ASCII codes and an example of packed 6-bit ASCII characters is included later in this section.

### G-3. BCD PLUS definition:

0h - 9h = digits 0 through 9  
 Ah = space  
 Bh = dash '-'  
 Ch = period '.'  
 Dh = reserved  
 Eh = reserved  
 Fh = reserved

### G-4. Product Info Area Format:

field length	field
1	Product Area Format Version 7:4 - reserved, write as 0000b 3:0 - format version number = 1h for this specification
1	Product Area Length (in multiples of 8 bytes)
1	Language Code ('0' or '25' decimal for English )
1	<b>Manufacturer Name</b> type/length byte
N	Manufacturer Name bytes
1	<b>Product Name</b> type/length byte
M	Product Name bytes
1	<b>Product Part/Model Number</b> type/length byte
O	Product Part/Model Number bytes
1	<b>Product Version</b> type/length byte
R	Product Version bytes
1	<b>Product Serial Number</b> type/length byte*
P	Product Serial Number bytes*
1	<b>Asset Tag</b> type/length byte
Q	Asset Tag
1	<b>FRU File ID</b> type/length byte
R	FRU File ID bytes. The FRU File version field is a pre-defined field provided as a manufacturing aid for verifying the file that was used during manufacture or field update to load the FRU information. The content is manufacturer-specific. This field is also provided in the Board Info area. Either or both fields may be 'null'.
xx	Custom product info area fields, if any (must be preceded with type/length byte)
1	C1h (type/length byte encoded to indicate no more info fields).
Y	00h - any remaining unused space
1	Product Info Area Checksum (zero checksum)

\* These fields are always encoded as if the Language Code were English. I.e. if the type/length code bits 7:6=11b the serial number will always be interpreted as ASCII+Latin 1, not UNICODE.

## Appendix H IPMI/IPMB Reference Documents

### References:

- 1) IPMI – Intelligent Platform Management Interface Specification, V1.0, Document Revision 1.1, November 15, 1999
- 2) IPMI – Intelligent Platform Management Bus Communications Protocol Specification, v1.0, Document Revision 1.0, November 15, 1999
- 3) Intelligent Platform Management Interface Implementers Guide, Draft – Version 0.7, 9/16/98
- 4) IPMI – Intelligent Platform Management Interface Specification, V1.5, Document Revision 1.0, February 21, 2001
- 5) Platform Management FRU Information Storage Definition, V1.0, Document Revision 1.1, September 27, 1999
- 6) IPMI – IPMB v1.0 Address Allocation, Document Revision 1.0, September 16, 1998.
- 7) IPMI – Intelligent Platform Management Interface Specification, V1.0, Document Revision 1.1, November 15, 1999
- 8) IPMI – Intelligent Platform Management Bus Communications Protocol Specification, v1.0, Document Revision 1.0, November 15, 1999
- 9) Intelligent Platform Management Interface Implementers Guide, Draft – Version 0.7, 9/16/98
- 10) IPMI – Intelligent Platform Management Interface Specification, V1.5, Document Revision 1.0, February 21, 2001
- 11) Platform Management FRU Information Storage Definition, V1.0, Document Revision 1.1, September 27, 1999
- 12) IPMI – IPMB v1.0 Address Allocation, Document Revision 1.0, September 16, 1998.