IBC 2501

CMC Chassis Management Controller

Programmer's Guide

Part No. 100543 Rev D

Trademarks
IBM PC is a registered trademark of International Business Machines Corporation. Intel and Pentium are registered trademarks of Intel Corporation. Award is a registered trademark of Award Software, Inc. Other product names mentioned herein are used for identification pur- poses only and may be trademarks and/or registered trademarks of their respective companies.

C ustomer Service

Worldwide Headquarters
I-Bus Corporation
2391 Zanker Road #380
San Jose
CA 95131, USA
Tel: +(1) 408 428 6100
Fax: +(1) 408 428 6101
Toll Free: 877-777-IBUS
Email: sales@ibus.com

European Headquarters
I-Bus
Unit 6, Chichester Business Park
City Fields Way, Tangmere
West Sussex, PO20 2LB, UK
Tel: +44 (0) 1243 756300
Fax: +44 (0) 1243 756301
Email: sales@ibus.co.uk

France, Italy
I-Bus
B.P 45 Valbonne
06901 Sophia Antipolis CEDEX
France
Tel: +33 (0) 493 004 360
Fax: +33 (0) 493 004 369
Email: sales@ibus.com

Scandinavia
I-Bus
Drakegatan 10 - BOX 184
401 23 Göteborg
Sweden
Tel: +46 (0)31-773 7117
Fax: +46 (0)31-773 7075
Email: pry@ibus.com

Thank You
from the

team for
purchasing this product

Dear Customer,

Thank you for purchasing an I-Bus Corporation product. We hope that this product exceeds your expectations. It is our desire to provide you with accurate, up-to-date information about the product(s) you have purchased. We welcome your comments and suggestions about our manuals.

You may email those comments and suggestions to support@ibus.com. Please be sure to include your name, the name of your company, the product you purchased, and the manual number/revision (i.e. 00-00000-00 Rev. *). This number is located on the title page.

At I-Bus Corp., we value our customers and partners, and you can continue to count on I-Bus to be customer focused and to provide you a large range of solutions -- from cost-effective to fully customized industrial computer solutions.

Again, thank you for your committment to I-Bus Corporation. We appreciate your business and look forward to continuing to work with you and helping you reach your goals.

Chapter 1. ***Introduction***

Chapter 2. ***IPMI Interface***

1 Introduction

1.1 Overview

Welcome to the I-Bus Chassis Management Controller (CMC).

This manual describes the functional and software interface specifications of the I-Bus CMC Board. The CMC Board functions to monitor and control the system environment of a custom cPCI enclosure. It operates as an independent Chassis Management Controller manageable through its RS232 port by an external device. It conforms to the PICMG 2.9 R1.0 System Management Specification, supporting the following:

IPMI Command Protocol
Local I^2C bus to connect temperature sensors and any other remote sensing devices

1.2 Description

The CMC Board is plugged into a special slot on the back of an I-Bus cPCI monolithic backplane.

The CMC board has two I^2 C ports, to support cPCI System Management Bus per PICMG 2.9. One of the ports is used as the local I^2C bus to be connected to temperature sensors, etc. The other port is used as the IPMB0 bus and routed to the cPCI backplane connectors.

Control and communication with the CMC board is via IPMI-command structure through the RS232 port(s). For example, the host controller is capable of changing and setting limits/masks for the sensors via the IPMI command.

The CMC board is capable of replying to messages from the host controller.

Chapter 1 - Introduction

This page was intentionally left blank

2 IPMI Interface

2.1 IPMI Communications for the CMC Board

This chapter describes the commands and communications for the CMC Board for use in the I-Bus C08xx and the C0406A system.

2.1.1 CMC IPMI COMMANDS AND CONFIGURATION

All specified completion codes and returned data are in hexadecimal. The completion codes 00h (Success), C7h (Request data length invalid), CCh (Invalid data field in request), and FFh (Unspecified Error) are always possible. Other codes are possible only where noted. If too many or too few parameters are supplied, the return value will be C7h. If parameters are supplied that are not listed as valid, CCh will be returned. Unsupported commands will return the completion code C1h.

Table of IPMI-based Addresses and IDs used by the CMC

ID	Description	Type
00h	Monitor chassis temperature	Temperature Sensors (°C)
01h		
02h		
03h		
04h – 07h		
08h	Monitor fan speeds	Fan speed sensors (RPM)
09h		
0Ah		
0Bh	Monitor fan health	Fan health sensors (On/Off)
0Ch		
0Dh		
0Eh	+3.3 volt	Voltage Sensors (millivolts)
0Fh	+5.0 volts	
10h	Battery volts	
11h	+12 volts	
12h	-12 volts	
15h	IPMB voltage	
13h	-48 volts A	Voltage Sensors (volts)
14h	-48 volts B	
64h	CMC Board acting as BMC	Slave Address
0	Local Serial EEPROM	I^2C Bus ID
1	IPMB0	
2	Time-of-day chip	
3	Local (Temperature sensors)	
18h 77h	C0877 CMC	Product ID
18h 14h	C0814 CMC	
14h 06h	C0406A CMC	
01h or 02h	01h for the 08XX version of the CMC and 02h for the 04XX version	Device ID
001167h	IANA assigned manufacturer ID for I-Bus	Manufacturer ID

2.1.2 Sensor Data Records

For your reference the following is a brief description of the default set of SDRs that are provided in the Sensor Data Record Repository. All data is presented in hexadecimal and briefly described. Please refer to IPMI Specification v1.0, Section 28 for the full description of SDRs.

2.1.2.1 Temperature Sensor SDRs

SDR Version: 01h, Version 1.0

Type: 01h, Full Sensor Records

Additional Length: 49 bytes.

Sensor Owner ID, LUN: 64h, LUN 00h

Sensor ID: 00h – 07h

Entity ID and Instance: Both are unspecified.

Sensor Initialization: 3Fh

- Initialize events, thresholds, hysteresis, and sensor type.
- Enable both scanning and event generation.

Sensor Capabilities: 64h

- Don't ignore sensor.
- Hysteresis and Thresholds are readable and settable.
- Event messages can be disabled on a per-sensor and per-threshold basis

Sensor Type: 01h, Temperature

Event / Reading Type Code: 01h, Threshold

Masks for Assertion Events and Lower threshold reading: 95h, 7Ah

- Events for upper thresholds going high and lower thresholds going low are supported
- All lower threshold comparisons are returned in Get Sensor Reading

Masks for Deassertion Events and Upper threshold reading: 95h, 7Ah

- Events for upper thresholds going high and lower thresholds going low are supported
- All upper threshold comparisons are returned in Get Sensor Reading

Masks for Readable and Settable Thresholds: 3Fh, 3Fh, All thresholds are readable and settable.

Sensor Units 1: 80h, 2's complement, signed.
Sensor Units 2: 01h, degrees Celsius.
Sensor Units 3: 00h, unused.
Linearization: 00h
M and Tolerance: 01h, 00h, M (resolution) is 1, tolerance is 0.
B and Accuracy: 00h, 00h, 00h, all are 0.
R and B exponents: 00h, both are 0.
Analog characteristics: 07h, nominal reading and normal max and min are specified.
Nominal Reading: 19h
Normal Min and Max: 14h – 1Eh
Sensor Min and Max reading: 80h – 7Fh
Default Threshold Settings:

- Upper non-recoverable: 32h
- Upper critical: 2Dh
- Upper non-critical: 28h
- Lower non-recoverable: 00h
- Lower critical: 0Ah
- Lower non-critical: 0Fh

Positive-going Hysteresis: 02h
Negative-going Hysteresis: 02h
ID String (ASCII): LM75#x where x is the sensor id.

2.1.2.2 Tach-Fan SDRs

SDR Version: 01h, Version 1.0
Type: 01h, Full Sensor Records
Additional Length: 48 bytes.
Sensor Owner ID, LUN: 64h, LUN 00h
Sensor ID: 08h – 0Ah
Entity ID and Instance: Both are unspecified.
Sensor Initialization: 3Fh

- Initialize events, thresholds, hysteresis, and sensor type.
- Enable both scanning and event generation.

Sensor Capabilities: 64h

- Don't ignore sensor.
- Hysteresis and Thresholds are readable and settable.
- Event messages can be disabled on a per-sensor and per-threshold basis.

Sensor Type: 04h, Fan

Event / Reading Type Code: 01h, Threshold

Masks for Assertion Events and Lower threshold reading: 95h, 7Ah

- Events for upper thresholds going high and lower thresholds going low are supported
- All lower threshold comparisons are returned in Get Sensor Reading

Masks for Deassertion Events and Upper threshold reading: 95h, 7Ah

- Events for upper thresholds going high and lower thresholds going low are supported
- All upper threshold comparisons are returned in Get Sensor Reading

Masks for Readable and Settable Thresholds: 3Fh, 3Fh, All thresholds are readable and settable.

Sensor Units 1: 00h, unsigned.

Sensor Units 2: 12h, RPM.

Sensor Units 3: 00h, unused.

Linearization: 00h, linear

M and Tolerance: 20h, 00h, M (resolution) is 20, tolerance is 0.

B and Accuracy: 00h, 00h, 00h, all are 0.

R and B exponents: 00h, both are 0.

Analog characteristics: 07h, nominal reading and normal max and min are specified.

Nominal Reading: 61h.

Normal Min and Max: 5Eh – 64h.

Sensor Min and Max reading: 00h - FFh.

Default Threshold Settings:

- Upper non-recoverable: 91h.
- Upper critical: 74h.
- Upper non-critical: 6Ah.
- Lower non-recoverable: 30h.
- Lower critical: 4Dh.
- Lower non-critical: 57h.

Positive-going Hysteresis: 03h.

Negative-going Hysteresis: 03h.

OEM Byte: 3Fh, all threshold violations should cause alarms.

ID String (ASCII): FAN#x where x is the sensor id minus 8.

2.1.2.3 Binary-Fan SDRs

SDR Version: 01h, Version 1.0

Type: 01h, Full Sensor Records

Additional Length: 48 bytes.

Sensor Owner ID, LUN: 64h, LUN 00h

Sensor ID: 08h – 0Ah

Entity ID and Instance: Both are unspecified.

Sensor Initialization: 3Fh

- Initialize events, thresholds, hysteresis, and sensor type.
- Enable both scanning and event generation.

Sensor Capabilities: 4Ch

- Don’t ignore sensor.
- No Hysteresis and Thresholds are fixed.
- Event messages can be disabled on a per-sensor and per-threshold basis.

Sensor Type: 04h, Fan

Event / Reading Type Code: 01h, Threshold

Masks for Assertion Events and Lower threshold reading: 80h, 00h

- Only upper non-critical going-high assertion event is generated*
- No threshold comparisons are returned in Get Sensor Reading

*All upper threshold events are supported, but some are turned off by default.

Masks for Deassertion Events and Upper threshold reading: 80h, 00h

- Only upper non-critical going-high deassertion event is generated*
- No threshold comparisons are returned in Get Sensor Reading

Masks for Readable and Settable Thresholds: 3Fh, 00h, No thresholds are settable.

Sensor Units 1: C0h, no analog reading.

Sensor Units 2: 00h, unused.

Sensor Units 3: 00h, unused.

Linearization: 00h, linear

M and Tolerance: 01h, 00h, M (resolution) is 1, tolerance is 0.

B and Accuracy: 00h, 00h, 00h, all are 0.

R and B exponents: 00h, both are 0.

Analog characteristics: 07h, nominal reading and normal max and min are specified.

Nominal Reading: 00h.

Normal Min and Max: 00h – 00h.

Sensor Min and Max reading: 00h - 01h.

Default Threshold Settings:

- Upper non-recoverable: 01h.
- Upper critical: 01h.
- Upper non-critical: 01h.
- Lower non-recoverable: 00h.
- Lower critical: 00h.
- Lower non-critical: 00h.

Positive-going Hysteresis: 00h.

Negative-going Hysteresis: 00h.

OEM Byte: 08h, upper non-critical threshold violation should cause alarm.

ID String (ASCII): FAN#x where x is the sensor id minus 8.

*All upper threshold events are supported, but some are turned off by default.

2.1.2.4 Voltage Sensors

SDR Version: 01h, Version 1.0

Type: 01h, Full Sensor Records

Additional Length: 49 bytes.

Sensor Owner ID, LUN: 64h, LUN 00h

Sensor ID: 0Eh – 15h

Entity ID and Instance: Both are unspecified.

Sensor Initialization: 3Fh

- Initialize events, thresholds, hysteresis, and sensor type.
- Enable both scanning and event generation.

Sensor Capabilities: 64h

- Don't ignore sensor.
- Hysteresis and Thresholds are readable and settable.
- Event messages can be disabled on a per-sensor and per-threshold basis.

Sensor Type and Event / Reading Type: 02h, 01h, Voltage, Threshold

Masks for Assertion Events and Lower threshold reading: 04h, 72h

- Upper critical going-high and lower critical going-low events are generated*
- All lower threshold comparisons are returned in Get Sensor Reading

Masks for Deassertion Events and Upper threshold reading: 04h, 72h

- Events for upper thresholds going high and lower thresholds going low are supported
- Upper critical going-high and lower critical going-low events are generated*

Masks for Readable and Settable Thresholds: 3Fh, 3Fh, All are readable and settable.

Sensor Units 1, 2, and 3: 00h, 04h, 00h, unsigned volts.

Linearization: 00h, linear

*All upper going-high and lower going-low events are supported, but some are turned off by default.

M, Tolerance, B, Accuracy, R exponent and B exponent:

Sensor	M	Tol.	B	Acc.	R exp.	B exp.	Raw data
14	103	0	198	0	-4	2	67h 00h C6h 00h 00h 2Ch
15, 21	156	0	300	0	-4	2	9Ch 00h 2Ch 40h 00h 2Ch
16	188	0	360	0	-4	2	BCh 00h 68h 40h 00h 2Ch
17	-376	0	-72	0	-4	3	88h 80h B8h C0h 00h 3Ch
18	376	0	72	0	-4	3	78h 40h 48h 00h 00h 3Ch
19, 20	-150	0	-288	0	-3	2	6Ah C0h E0h 80h 00h 2Dh

Analog characteristics: 07h, nominal reading and normal max and min are specified.

Sensor	Nominal	Min	Max	Sens. Min	Max	Raw data
14	80h	7Bh	85h	00h	FFh	80h 7Bh 85h 00h FFh
15, 17, 18, 21	80h	7Ah	86h	00h	FFh	80h 7Ah 86h 00h FFh
16	80h	78h	8Ah	00h	FFh	80h 78h 8Ah 00h FFh
19, 20	80h	7Dh	83h	00h	FFh	80h tDh 83h 00h FFh

Default Threshold Settings:	Upper Thresholds	Lower Thresholds
Sensor 16	9Ah	65h
Sensors 14, 15, 17, 18, 21	90h	70h
Sensors 19 and 20	83h	7Dh

Positive- and Negative-going Hysteresis: 00h, 00h

ID String (ASCII): Volt#x where x is the sensor id minus 14.

OEM byte: 12h, Critical Threshold violations should cause alarms.

2.1.3 IPMB Address

The IPMB address is factory set to 64h.

2.1.4 IPMI Commands

The following pages give the IPMI commands that are supported by the CMC.

Under **Syntax**, the SEQ value can be anything. It is used, as described in IPMI, to match responses with requests.

2.1.4.1 IPMI Cold Reset Command

The Cold Reset command resets the CMC as if it were turned off and back on again. The effects of a Cold Reset are as follows:

- Existing alarm conditions will cause events to be added to the SEL.
- The ACO will become inactive.
- Any reservation of the SEL will be cancelled.
- All sensor thresholds and hysteresis will be reset to their default values.
- Any reservation of the SDR will be cancelled.
- An internal self-test will be performed.

IPMI Cold Reset Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	18h	02h	02h

	Byte	**Data Field**
Request Data	-	-
Response Data	-	-

2.1.4.2 IPMI Get Self Test Results Command

The Get Self Test Results command returns the result of a check for the availability of the SEL and SDR logical devices.

IPMI Get Self Test Results Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	18h	03h	04h

	Byte	Data Field
Request Date	-	-
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2	55h = No errors 57h = Corrupted or inaccessible devices (see below)
	3	00h = No errors 08h = SDR Repository was empty, now restored to manufacturing default E0h = SDR and SEL access is faulty E8h = Both of the above problems were detected

2.1.4.3 IPMI Get SEL Info Command

The Get SEL Info command returns information about the number of entries in the SEL and the time stamp of the last addition to the SEL.

IPMI Get SEL Info Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	28h	14h	40h

	Byte	Data Field
Request Data	-	-
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2	Indicates the SEL version, which is 01h
	3	Number of entries in SEL, LS Byte
	4	Number of entries in SEL, MS Byte
	5	Free Space in bytes, LS Byte
	6	Free Space in bytes, MS Byte
	7-10	Timestamp of most recent addition to the SEL
	11-14	Most recent erase timestamp. Last time that one or more entries were deleted from SEL.
	15	Operation Support bit 7-4: 0 bit 3: 1=Delete SEL command supported bit 2; 1=Partial Add SEL Entry command supported bit 1: 1=Reserve SEL command supported bit 0: 1=Get SEL Allocation Information command supported

2.1.4.4 IPMI Reserve SEL Command

The Reserve SEL command prepares to signal the caller if the SEL changes before the caller is done using it. If the SEL changes, the caller receives a C5h (Reservation cancelled) completion code when using commands that require the SEL Reservation ID.

IPMI Reserve SEL Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	28h	15h	42h

	Byte	Data Field
Request Data	-	-
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2	Reservation ID, LS Byte 0000h reserved
	3	Reservation ID, MS Byte

2.1.4.5 IPMI Get SEL Entry Command

The Get SEL Entry command returns the event record data for a specified event.

IPMI Get SEL Entry Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
09h	28h	16h	43h	ResID, RecID, Offset, Count

	Byte	Data Field
Request Data	1	Reservation ID, LS Byte. Only required for partial Get. Use 0000h otherwise.
	2	Reservation ID, MS Byte
	3	SEL Record ID, LS Byte
	4	SEL Record ID, MS Byte 0000h = GET FIRST ENTRY FFFFh = GET LAST ENTRY
	5	Offset into record
	6	Bytes to read. FFh means read entire record.
Response Data	1	Completion code 00h = Command completed normally C5h = Reservation cancelled C7h = Request data length invalid C9h = Parameter out of range CAh = Cannot return number of requested data bytes CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error
	2	Next SEL Record ID, LS Byte
	3	Next SEL Record ID, MS Byte
	4-19	Record Data, 16 bytes

2.1.4.6 IPMI Clear SEL Command

The Clear SEL command erases all records from the SEL or checks on the progress of a previous request.

IPMI Clear SEL Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
09h	28h	18h	47h	ResID, CLR, Status

	Byte	Data Field
Request Data	1	Reservation ID, LS Byte (this is the 2nd byte returned from Reserve SEL)
	2	Reservation ID, MS Byte (this is the 3rd byte returned from Reserve SEL)
	3	'C' (43h)
	4	'L' (4Ch)
	5	'R' (52h)
	6	AAh = Initiate erase 00h = Get erasure status
Response Data	1	Completion Code 00h = Command completed normally C5h = Reservation cancelled C7h = Request data length invalid CCh = Invalid data field in request FFh = Unspecified error
	2	Erasure progress 00h = Erasure in progress 01h = Erase completed

2.1.4.7 IPMI Get SEL Time Command

The Get SEL Time command returns the current value in the SEL clock. Until the SEL clock is set using the *Set SEL Time* command, the timestamps used will start at 00000000h, IPMI specifies that values below 20000000h be interpreted as "seconds since the controller was last powered up."

IPMI Get SEL Time Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	28h	19h	48h

	Byte	Data Field
Request Data	-	-
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2-5	Present Timestamp clock reading, LS Byte first

2.1.4.8 IPMI Set SEL Time Command

The Set SEL Time command sets the SEL clock.

IPMI Set SEL Time Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
07h	28h	1Ah	49h	Time

	Byte	Data Field
Request Data	1-4	Time in four-byte format, LS Byte first
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CCh = Invalid data field in request FFh = Unspecified error

2.1.4.9 IPMI Delete SEL Entry Command

The Delete SEL Entry Command deletes a specified record from the SEL. Since this is a sensitive operation, it will cancel the current SEL Reservation ID.

IPMI Delete SEL Entry Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
07h	28h	1Bh	46h	ResID, Rec ID

	Byte	Data Field
Request Data	1	Reservation ID, LS Byte (this is the 2nd byte returned from Reserve SEL)
	2	Reservation ID, MS Byte (this is the 3rd byte returned from Reserve SEL)
	3	SEL Record ID to delete, LS Byte
	4	SEL Record ID to delete, MS Byte 0000h = FIRST ENTRY FFFFh = LAST ENTRY
Response Data	1	Completion Code 00h = Command completed normally C5h = Reservation cancelled C7h = Request data length invalid CBh = Record not present CCh = Invalid data field in request FFh = Unspecified error plus the following command specific: 80h = Operation not supported for this Record Type
	2	Record ID for deleted Record, LS Byte
	3	Record ID for deleted Record, MS Byte

2.1.4.10 IPMI Get SDR Repository Info Command

The Get SDR Repository Info command returns information about the SDR including timestamps, the version of the SDR and the number of records it holds.

IPMI Get SDR Repository Info Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	28h	1Ch	20h

	Byte	Data Field
Request Data	-	-
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2	Indicates the SDR version, which is 01h
	3	Record count LS Byte - number of records in the SDR Repository
	4	Record count MS Byte - number of records in the SDR Repository
	5-6	Free space in bytes, LS Byte first. 0000h indicates 'full'. If more than 65,534 bytes are free, the entry here should be FFFEh. FFFFh indicates 'unspecified'.
	7-10	Most recent addition timestamp, LS Byte first.
	11-14	Most recent erase timestamp, LS Byte first
	15	Operation Support bit 7-4: 0 bit 3: 1=Delete SDR command supported bit 2: 1=Partial Add SDR command supported bit 1: 1=Reserve SDR Repository command supported bit 0: 1=Get SDR Allocation information command supported

2.1.4.11 IPMI Reserve SDR Repository Command

The Reserve SDR Repository command prepares to signal the caller if the SDR changes before the caller is done using it. If the SDR changes, the caller receives a C5h (Reservation cancelled) completion code when using commands that require the SDR Reservation ID.

IPMI Reserve SDR Repository Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	28h	1Dh	22h

	Byte	Data Field
Request Data	-	-
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2	Reservation ID, LS Byte
	3	Reservation ID, MS Byte

2.1.4.12 IPMI Get SDR Command

The Get SDR command returns the sensor data record for the specified record ID.

IPMI Get SDR Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
09h	28h	16h	23h	ResID, RecID, Offset, Count

	Byte	Data Field
Request Data	1	Reservation ID, LS Byte (this is the 2nd byte returned from Reserve SDR). Only required for partial reads with a non-zero 'Offset into record' field. Otherwise you may use 0000h for reservation ID.
	2	Reservation ID, MS Byte (this is the 3rd byte returned from Reserve SDR)
	3	Record ID of record to Get, LS Byte. To get the first record use 00h, and to get the last record use FFh.
	4	Record ID of record to Get, MS Byte. To get the first record use 00h, and to get the last record use FFh.
	5	Indicates the offset into record if only a specific sub-section of the record is needed. If this byte is greater than the length of the record, the completion code will be C9h
	6	Indicates how many bytes are being requested. To request all bytes, use FFh. If there are not enough bytes in the record to fill this request or if more than 16 bytes are requested, the completion code will be CAh.

Response Data	1	Completion Code 00h = Command completed normally C5h = Reservation cancelled C7h = Request data length invalid C9h = Parameter out of range CAh = Cannot return number of requested data bytes CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error
	2	Record ID for next Record, LS Byte
	3	Record ID for next Record, MS Byte
	4-3+ N	Record Data

2.1.4.13 IPMI Partial Add SDR Command

The Partial Add SDR command adds a record or a piece of a record to the SDR.

IPMI Partial Add SDR Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
0Ah - 19h	28h	1Fh	25h	ResID, RecID, Offset, Last, Data

	Byte	Data Field
Request Data	1	Reservation ID, LS Byte. Only required for partial add. Use 0000h for reservation ID otherwise.
	2	Reservation ID, MS Byte
	3	Record ID, LS Byte for continuing partial add. Use 0000h for Record ID otherwise.
	4	Record ID, MS Byte for continuing partial add. Use 0000h for Record ID otherwise.
	5	Indicates the offset into the record at which to add the data
	6	In progress. 00h = partial add in progress. 01h = last record data being transferred with this request
	7-N	SDR Record Data
Response Data	1	Completion Code 00h = Command completed normally C4h = Out of space C5h = Reservation cancelled C7h = Request data length invalid C9h = Parameter out of range CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error
	2	Record ID for added Record, LS Byte
	3	Record ID for added Record, MS Byte

2.1.4.14 IPMI Delete SDR Command

The Delete SDR command deletes a record from the SDR Repository.

IPMI Delete SDR Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
07h	28h	20h	26h	ResID, RecID

	Byte	Data Field
Request Data	1	Reservation ID, LS Byte
	2	Reservation ID, MS Byte
	3	Record ID of record to delete, LS Byte
	4	Record ID of record to delete, MS Byte
Response Data	1	Completion Code 00h = Command completed normally C5h = Reservation cancelled C7h = Request data length invalid CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error
	2	Record ID for deleted Record, LS Byte
	3	Record ID for deleted Record, MS Byte

2.1.4.15 IPMI Get Sensor Reading Factors Command

The Get Sensor Reading Factors command retrieves data that can be used to calculate a human-readable value from the raw output of the sensor. The command assumes that the SMC contains one table for each sensor. Each row in the table contains factors and a raw reading value. The range of readings for which the factors are intended is from the raw reading in that row to the raw reading in the next row (or to FFh if it's the last row). The formula to convert the raw reading (X) to a human readable one (Y) is:

$$Y = (mX + b \times 10^{k_1}) \times 10^{k_2}$$

m, b, k_1, and k_2 are specified in the table, as well as a 10-bit accuracy and a 6-bit tolerance in 0.5 raw counts.

IPMI Get Sensor Reading Factors Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
05h	10h	25h	23h	ID, Reading

	Byte	Data Field
Request Data	1	Sensor number (00h - 15h)
	2	Indicates the reading to which these factors will be applied

Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CCh = Invalid data field in request FFh = Unspecified error
	2	If the submitted reading matches one in the table, this byte indicates the next reading in the table. If the submitted raw reading didn't match a reading in the table, this returns the nearest reading that is in the table.
	3	Indicates the least significant 8 bits of M
	4	bits 7 and 6 are the most significant bits of M, and bits 5-0 indicate the tolerance in ±0.5 raw counts
	5	Indicates the least significant bits of B
	6	bits 7 and 6 are the most significant bits of B, and bits 5-0 are the least significant bits of the accuracy.
	7	bits 7-4 are the most significant bits of the accuracy, and bits 3 and 2 are the unsigned accuracy exponent
	8	bits 7-4 are K2 and bits 3-0 are K1

2.1.4.16 IPMI Get Sensor Threshold Command

The Get Sensor Threshold command returns the thresholds associated with the specified sensor.

IPMI Get Sensor Threshold Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
04h	10h	26h	27h	ID

	Byte	Data Field
Request Data	1	Sensor number (00h - 15h)
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error
	2	The value 3Fh shall be returned to indicate that all thresholds are being returned.
	3	The lower non-critical threshold for this sensor
	4	The lower critical threshold for this sensor
	5	The lower non-recoverable threshold for this sensor
	6	The upper non-critical threshold for this sensor
	7	The upper critical threshold for this sensor
	8	The upper non-recoverable threshold for this sensor

2.1.4.17 IPMI Set Sensor Threshold Command

The Set Sensor Threshold command sets the thresholds compared to the current reading of this sensor to generate threshold events. Not all thresholds have to be set at once. If a threshold is not to be set, it can be omitted, as long as any thresholds that normally come after it are also omitted. However, if it is not omitted, 00h should be used as the parameter value.

IPMI Set Sensor Threshold Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
06h - 0Bh	10h	27h	26h	ID, Thresholds, Threshold Data

	Byte	Data Field
Request Data	1	Sensor number (00h - 15h)
	2	bit 7-6: 0 bit 5: 1-set upper non-recoverable threshold bit 4: 1-set upper critical threshold bit 3: 1-set upper non-critical threshold bit 2: 1-set lower non-recoverable threshold bit 1: 1-set lower critical threshold bit 0: 1-set lower non-critical threshold
	3	lower non-critical threshold
	4	lower critical threshold
	5	lower non-recoverable threshold
	6	upper non-critical threshold
	7	upper critical threshold
	8	upper non-recoverable threshold
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error

2.1.4.18 IPMI Set Sensor Hysteresis Command

The Set Sensor Hysteresis command tells the CMC how far back from a threshold the sensor readings must come before another event can be triggered for crossing that threshold.

IPMI Set Sensor Hysteresis Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
07h	10h	28h	24h	ID, FF, Pos, Neg

	Byte	Data Field
Request Data	1	Sensor number (00h - 15h)
	2	FFh (as specified by IPMI)
	3	Positive going threshold hysteresis value. This is used when deasserting threshold-going-high states.
	4	Negative going threshold hysteresis value. This is used when deasserting threshold-going-low states.
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error

2.1.4.19 IPMI Get Sensor Hysteresis Command

The Get Sensor Hysteresis command indicates the current hysteresis values for the specified sensor.

IPMI Get Sensor Hysteresis Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
05h	10h	29h	25h	ID, FF

	Byte	Data Field
Request Data	1	Indicates the sensor ID (00h - 15h)
	2	FFh (as specified by IPMI)
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error
	2	Positive going threshold hysteresis value. This is used when deasserting threshold-going-high states.
	3	Negative going threshold hysteresis value. This is used when deasserting threshold-going-low states.

2.1.4.20 IPMI Get Sensor Reading Command

The Get Sensor Reading command returns the current reading of the specified sensor and what thresholds it is at or beyond, if any, regardless of hysteresis settings.

IPMI Get Sensor Reading Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
04h	10h	2Ah	2Dh	ID

	Byte	Data Field
Request Data	1	Indicates the sensor ID (00h - 15h)

<table>
<tr><td rowspan="8">Response Data</td><td>1</td><td colspan="2">Completion Code
00h = Command completed normally
C7h = Request data length invalid
CBh = Requested sensor, data, or record not present
CCh = Invalid data field in request
FFh = Unspecified error</td></tr>
<tr><td>2</td><td colspan="2">The output of the sensor</td></tr>
<tr><td>3</td><td colspan="2">bit 7: 0=this sensor will not generate events
bit 6: 0=this sensor is not being scanned by the firmware
bit 5: 1=the SMC is initializing and the threshold comparison
data may not be valid
bit 4-0:
0</td></tr>
<tr><td rowspan="7">4</td><td colspan="2">This is a bit field that indicates what thresholds sensor reading is at or beyond.</td></tr>
<tr><td>If this bit is set:</td><td>The reading is at or beyond this threshold:</td></tr>
<tr><td>0</td><td>Lower non-critical</td></tr>
<tr><td>1</td><td>Lower critical</td></tr>
<tr><td>2</td><td>Lower non-recoverable</td></tr>
<tr><td>3</td><td>Upper non-critical</td></tr>
<tr><td>4</td><td>Upper critical</td></tr>
<tr><td>5</td><td>Upper non-recoverable</td></tr>
<tr><td>5</td><td colspan="2">00h (Reserved for discrete sensors)</td></tr>
</table>

2.1.4.21 IPMI Set Sensor Event Enable Command

The Set Sensor Event Enable command controls which threshold events generate event messages or SEL entries. The Amask and Dmask parameters are not required, but if they are missing they are interpreted to be 00h.

If this command is used to disable upper-threshold going-high events or lower-threshold going-low events, the corresponding alarm states will not be changed either. This guarantees that if an alarm is on because of a threshold violation, then there will be an event for it.

IPMI Set Sensor Event Enable Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
05h - 09h	10h	30h	28h	ID, Control, Amask, Dmask

	Byte	Data Field
Request Data	1	Indicates the sensor ID (00h - 15h)
	2	bit 7: 0=disable all Event Messages from this sensor (optional) (does not impact individual enable/disable status) bit 6: 0=disable scanning on this sensor (optional) bit 5-4: 00b=do not change individual enables 01b=enable selected event messages 10b=disable selected event messages 11b=reserved bit 3-0: reserved
	3	Indicates which of certain assertion events are generated. If the indicated bit is 1b, an event will be generated when the indicated state is asserted: bit 7: 1=select assertion event for upper non-critical going high bit 6: the setting of this bit will have no effect because this event is not supported by the SMC bit 5: the setting of this bit will have no effect because this event is not supported by the SMC bit 4: 1=select assertion event for lower non-recoverable going low bit 3: the setting of this bit will have no effect because this event is not supported by the SMC bit 2: 1=select assertion event for lower critical going low bit 1: the setting of this bit will have no effect because this event is not supported by the SMC bit 0: 1=select assertion event for lower non-critical going low

Request Data	4	Indicates which of certain assertion events are generated. If the indicated bit is 01b, an event will be generated when the indicated state is asserted: bit 7-4: reserved; must be 0 bit 3: 1=select assertion event for upper non-recoverable going high bit 2: the setting of this bit will have no effect because this event is not supported by the SMC bit 1: 1=select assertion event for upper critical going high bit 0: the setting of this bit will have no effect because this event is not supported by the CMC

Request Data	5	Indicates which of certain deassertion events are generated. If the indicated bit is 01b, an event will be generated when the indicated state is deasserted: bit 7: 1=select deassertion event for upper non-critical going high bit 6: the setting of this bit will have no effect because this event is not supported by the CMC bit 5: the setting of this bit will have no effect because this event is not supported by the CMC bit 4: 1=select deassertion event for lower non-recoverable going low bit 3: the setting of this bit will have no effect because this event is not supported by the CMC bit 2: 1=select deassertion event for lower critical going low bit 1: the setting of this bit will have no effect because this event is not supported by the CMC bit 0: 1=select deassertion event for lower non-critical going low

Request Data	6	Indicates which of certain deassertion events are generated. If the indicated bit is 01b, an event will be generated when the indicated state is deasserted: bit 7-4: reserved; must be 0 bit 3: 1=select deassertion event for upper non-recoverable going high bit 2: the setting of this bit will have no effect because this event is not supported by the CMC bit 1: 1=select deassertion event for upper critical going high bit 0: the setting of this bit will have no effect because this event is not supported by the CMC
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error

2.1.4.22 IPMI Get Sensor Event Enable Command

The Get Sensor Event Enable command returns an indication of which threshold events generate events or SEL entries.

IPMI Get Sensor Event Enable Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
04h	10h	31h	29h	ID

	Byte	Data Field
Request Data	1	Indicates the sensor ID (00h - 15h)
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CBh = Requested sensor, data, or record not present CCh = Invalid data field in request FFh = Unspecified error
	2	bit 7: 0=All Event Messages disabled from this sensor bit 6: 0=Sensor scanning disabled bit 5-0: reserved. Ignore on read

Response Data	3	bit 7: 1=assertion event for upper non-critical going high enabled bit 6: the setting of this bit will have no effect because this event is not supported by the CMC bit 5: the setting of this bit will have no effect because this event is not supported by the CMC bit 4: 1=assertion event for lower non-recoverable going low enabled bit 3: the setting of this bit will have no effect because this event is not supported by the CMC bit 2: 1=assertion event for lower critical going low enabled bit 1: the setting of this bit will have no effect because this event is not supported by the CMC bit 0: 1=assertion event for lower non-critical going low enabled
	4	bit 7-4: reserved; must be 0 bit 3: 1=assertion event for upper non-recoverable going high enabled bit 2: the setting of this bit will have no effect because this event is not supported by the CMC bit 1: 1=assertion event for upper critical going high enabled bit 0: the setting of this bit will have no effect because this event is not supported by the CMC

Response Data	5	bit 7: 1=deassertion event for upper non-critical going high enabled bit 6: the setting of this bit will have no effect because this event is not supported by the CMC bit 5: the setting of this bit will have no effect because this event is not supported by the CMC bit 4: 1=deassertion event for lower non-recoverable going low enabled bit 3: the setting of this bit will have no effect because this event is not supported by the CMC bit 2: 1=deassertion event for lower critical going low enabled bit 1: the setting of this bit will have no effect because this event is not supported by the CMC bit 0: 1=deassertion event for lower non-critical going low enabled
	6	bit 7-4: reserved; must be 0 bit 3: 1=deassertion event for upper non-recoverable going high enabled bit 2: the setting of this bit will have no effect because this event is not supported by the CMC bit 1: 1=deassertion event for upper critical going high enabled bit 0: the setting of this bit will have no effect because this event is not supported by the CMC

2.1.4.23 IPMI Get Device ID Command

The Get Device command retrieves device revision information and indicates what IPM Devices are supported.

IPMI Get Device Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	18h	01h	01h

	Byte	Data Field
Request Data	-	-

Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2	Device ID. This number identifies the unique set of functionality provided by this BMC. For the SMC, this is 01h or 02h
	3	Device Revision: Bit 7 = 0 because the CMC does not provide Device SDRs. Bits 6 - 4 are reserved and therefore 0, and bits 3 - 0 indicate the revision of the SMC (0001b)
	4	Firmware Revision 1: Bit 7 = 1 indicates that the device is initializing or updating firmware and therefore not available for normal operation. Bits 6 - 0 indicate major revision of the firmware (000001b)
	5	Firmware Revision 2: Indicates the minor version of the firmware, BCD encoded (00h)
	6	IPMI version. 01h indicates version 1.0
	7	Device support bit field = 1Fh, indicating support for: IPMB Event Receiver FRU Inventory Device SEL Device SDR Repository Device Sensor Device
	8-10	Manufacturer ID = 4455, LSB first: 67h 11h 00h
	11-12	Product ID = varies - see table of IPMI-based Addresses and IDs.

2.1.4.24 IPMI Get Chassis Status Command

The Get Chassis Status command returns information regarding the high-level status of the system chassis and main power subsystem.

IPMI Get Chassis Status Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	00h	2Bh	01h

	Byte	Data Field
Request Data	-	-

Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2	Indicates the current power status. bit 7: 0 bit 6: 0 bit 5: 1 bit 4: 1 indicates a failed attempt to enter power-on or power-off state bit 3: 1 indicates that a power fault has been detected bit 2: 0 bit 1: 0 bit 0: 1 indicates that the chassis power is on.
	3	Indicates the current power status: bit 7-5: 0 bit 4: 1 indicates last power-on state was entered via an IPMI command bit 3: 1 indicates last power down was caused by a power fault bit 2: 0 bit 1-0 0 (not implemented)
	4	Indicates current power status bit 7-4: 0 bit 3: 1 indicates that a cooling device fault was detected bit 2: 0 bit 1-0: 0 (not implemented)

2.1.4.25 IPMI Add SEL Entry Command

The Add SEL Entry command adds an event record to the SEL.

IPMI Add SEL Entry Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameters
13h	28h	17h	44h	RecID, Type, Data

	Byte	Data Field
Request Data	1 - 2	These two bytes serve only as a placeholder for the record ID that the CMC will generate
	3	Type: Indicates the record Type: 02h System event record C0h - DFh OEM-defined, but bytes 4-7 will be replaced with the current timestamp by the CMC E0h - FFh OEM-defined
	4 - 16	These are the remaining 13 bytes of the record data. If the Type is 02h, then it must be formatted as described in table 19-1 of the IPMI specification.
Response Data	1	Completion Code 00h = Command completed normally C4h=Out of space C5h=Reservation cancelled C7h = Request data length invalid FFh = Unspecified error
	2 - 3	Indicate the newly created Record ID, LSB first

2.1.4.26 IPMI Clear SDR Repository Command

The Clear SDR Repository command erases all records from the SDR or checks on the progress of a previous request

IPMI Clear SDR Repository Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameters
09h	28h	21h	27h	ResID, CLR, Status

	Byte	Data Field
Request Data	1	Reservation ID, LS Byte
	2	Reservation ID, MS Byte
	3	'C' (43h)
	4	'L' (4Ch)
	5	'R' (52h)
	6	Indicates whether to clear the SEL or check the status of a previous request to clear it. AAh = clear SEL 00h = check on the status of a previous request
Response Data	1	Completion Code 00h = Command completed normally C5h=Reservation cancelled C7h = Request data length invalid FFh = Unspecified error
	2	01h if the erasure is complete; otherwise 00h

2.1.5 OEM Commands

I-Bus has chosen to use Network Function code 32h for the alarm system implemented in the CMC. This network function code is to be used in all IPMI-based I-Bus management controllers that control alarms.

Alarms can be turned on by sensor events or by the *Set Alarms* command. Following is a description of situations in which the command has been used to manipulate an alarm. In most cases, using *Set Alarms* will produce an entry in the SEL as described below.

Turning an alarm on

Using *Set Alarms* to turn an alarm on produces a "virtual condition" that warrants the alarm. The condition will persist until *Set Alarms* is used to turn the alarm off or the CMC's power is cycled or a *Cold Reset* is performed. Whether or not the alarm was on prior to the execution of *Set Alarms*, an entry will be added to the SEL. This provides the user with a record of the cause of the alarm if all sensor-related conditions for it are eliminated.

Turning an alarm off

When *Set Alarms* is used to turn an alarm off, the alarm is guaranteed to go off. However, as soon as an appropriate sensor condition is asserted, it will come back on. Note that such assertion involves hysteresis logic. This effect is similar to that of pressing the ACO button, but instead of controlling the audible alarm, it controls the alarm itself. If the alarm was on before this command, an entry will be added to the SEL. This provides the user with a record of why the alarm is off if previous events indicate that it should be on. If the alarm was already off, no entry will be added.

Once an alarm has been manipulated by the *Set Alarms* command, it will not reflect sensor logic. If the alarm was turned on by the command, two events must occur for the alarm to once again reflect sensor logic. First, the "virtual condition" must be removed through the execution of *Set Alarms* to turn the alarm off. Second, a sensor threshold violation must occur to turn it back on. Without this second event, the sensor may be off even though threshold violations are present. If the alarm was turned off by the command, only the second event is required to make the alarm reflect sensor logic. To restore all alarms to states that reflect sensor logic immediately, the *Cold Reset* command can be used.

SEL Entry format

When use of the *Set Alarms* command generates an entry in the SEL, it will have this format:

Byte	Field	Description
1:2	Record ID	ID used for SEL access.
3	Type	C0h, I-Bus Alarm Control Event.
4:7	Timestamp	Time when *Set Alarms* command was received.
8:10	Manufacturer ID	(see *Get Device ID* command for definition)
11	SetMask/LUN	Bits 7:2 record the parameter passed to *Set Alarms*. Bits 1:0 record the LUN on which it was executed.
12	Slave Address	The 7-bit slave address (bit 0 is 0b) of the CMC.
13	State Before	Byte 2 of output from *Get Alarms* as if it had been executed before *Set Alarms.*
14	State After	Byte 2 of output from *Get Alarms* as if it were executed after *Set Alarms.*
15	Firmware Rev 1	The major version number of the firmware
16	Firmware Rev 2	The minor version number of the firmware

2.1.5.1 Set Alarms Command

The Set Alarms command manipulates the critical, major, and minor alarms.

Set Alarms Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameter(s)
04h	C8h	2Dh	01h	SetMask

<table>
<tr><td></td><td>Byte</td><td colspan="2">Data Field</td></tr>
<tr><td>Request Data</td><td>1</td><td colspan="2">SetMask: Indicates how to manipulate each alarm. Unused bits must be set to 0. A bit pattern of two bits determines what happens to the alarm:</td></tr>
<tr><td></td><td></td><td>Bits</td><td>Alarm</td></tr>
<tr><td></td><td></td><td>7:6</td><td>Critical</td></tr>
<tr><td></td><td></td><td>5:4</td><td>Major</td></tr>
<tr><td></td><td></td><td>3:2</td><td>Minor</td></tr>
<tr><td></td><td></td><td>1:0</td><td>Must be 00b</td></tr>
<tr><td></td><td></td><td>Bit-pattern</td><td>Control the alarm like this</td></tr>
<tr><td></td><td></td><td>00b</td><td>Leave the alarm alone</td></tr>
<tr><td></td><td></td><td>01b</td><td>Toggle the alarm</td></tr>
<tr><td></td><td></td><td>10b</td><td>Turn the alarm off</td></tr>
<tr><td></td><td></td><td>11b</td><td>Turn the alarm on</td></tr>
<tr><td>Response Data</td><td>1</td><td colspan="2">Completion Code
00h = Command completed normally
C7h = Request data length invalid
CBh = Requested sensor, data, or record not present
CCh = Invalid data field in request
FFh = Unspecified error</td></tr>
</table>

2.1.5.2 Get Alarms Command

The Get Alarms command returns the states of the three alarms.

Get Alarms Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	C8h	2Eh	02h

	Byte	Data Field	
Request Data	-	-	
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error	
	2	Indicates the state of the three alarms. Each alarm will be in one of four logical states:	
		Bit-pattern	State
		00b	The alarm is off
		01b	The alarm was turned off by a *Set Alarms* command
		10b	The alarm is on because of a threshold violation
		11b	The alarm was turned on by a *Set Alarms* command. If the alarm was already on, the bit pattern will still be 11b.
		The following rows show which bits of byte 2 are used for which alarm:	
		7:6	Critical
		5:4	Major
		3:2	Minor
		1:0	(Will be set to 0)

2.1.5.3 Enter Firmware Update Mode Command

The Enter Firmware Update command places the CMC into firmware update mode. While in firmware update mode, the CMC will not accept IPMI commands through the RS-232 ports. The CMC may not return a completion code. After that, it will reset itself and run the new firmware.

Enter Firmware Update Command

• Syntax

Length	NetFn/LUN	SEQ	Command	Parameters
07h	20h	2Fh	01h	IBKey

	Byte	Data Field
Request Data	IBKey: This four-byte code is intended to prevent careless or malicious SMS from trying to update the firmware. In order of transmission, the key is:	
	1	11h
	2	67h
	3	DAh
	4	A5h
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid CCh=Invalid data field in request FFh = Unspecified error

2.1.5.4 Chassis Reset Command

The Chassis Reset command instructs the CMC to force the system host to reset the chassis. The CMC will force this reset either by cycling power to the chassis or by toggling the reset line in the backplane.

Chassis Reset Command

• Syntax

Length	NetFn/LUN	SEQ	Command
03h	D0h	2Bh	01h

	Byte	Data Field
Request Data	-	-
Response Data	1	Completion Code 00h = Command completed normally C7h = Request data length invalid FFh = Unspecified error
	2	02h if the power was cycled, or 03h if the reset line was toggled

This page was intentionally left blank

3 Introduction to the I-Bus CMC IPMI Command Set API

This chapter describes the CMC IPMI API software library provided by I-Bus. The library itself is a shared object that is installed in /usr/lib and its name is libibusipmi.so. In order to use this API library, you must include smc_ipmi_cmd.h and port.h. Your application must then be compiled using the –libusipmi option.

The IPMI command set is implemented in two layers. The layer closest to the OS is the port layer and the layer closest to the application and above the port layer is the IPMI layer. The IPMI layer depends on the port layer and uses the port layer to communicate with the CMC through the serial port.

The port layer is initialized by calling **portNew** with the full path name of the device that the SMC is connected to. Then the port handle that is returned from **portNew** is used in the call to **SmcIpmiCmdInitialize()** to initialize the IPMI command layer. The port layer contains a lock mechanism to protect against anyone else using the device that the CMC is connected to. This means that if Kermit, or another instance of the IPMI command API were to attempt to use the port, then access would be denied. This also means that the I-Bus SNMP agent **cannot** be used at the same time as this API.

Once the port layer and the IPMI layer are initialized, communication can begin. All IPMI commands are essentially wrappers for the IPMI commands that are described in Chapter 2 of this document. For more detailed information regarding these commands, please refer to this documentation. Each IPMI command takes 1 or more arguments. The last argument to each command is a response buffer. This is a character byte string containing the complete reply packet, in response to issuing that command to the CMC. The details of this response packet will not be discussed here as it is discussed in detail elsewhere in this document. Also, the details of the other arguments will not be discussed completely here as they are also documented in Chapter 2 this document.

Further information on the details of these commands can be found in Chapter 2 of this manual and in the IPMI specification itself.

Appendix 1 shows a complete sample program that demonstrates the use of the IPMI API.

3.1 THE PORT LAYER

This layer sets up and tears down the serial port communication between the application and the CMC board.

3.1.1 portNew()

This function initializes the Port layer. This must be the first Port layer function that is called by the application. Use the PortT handle as the input to the initialization of the IPMI layer.

Prototype

```
PortT* portNew (
     const char* device_name
);
```

Parameters

device_name – [in] name of the tty port through which communication is to take place.

Return Value

If the command is successful, it returns a pointer to a handle that represents the connection to the serial port device.

If the command is unsuccessful, a NULL pointer is returned. It may be NULL if it could not gain exclusive access to the serial port. It may also be null if the port does not exist, or there were permission problems in connecting to the port. The lock file resides in /var/spool/lock, so the application must have the correct permissions for access to this directory. The application must also have access rights to the serial port.

Remarks

None.

3.1.2 portDelete()

This function breaks down the port layer and should be called when the application is exiting. This also removes the lock file.

Prototype

```
void portDelete (
     PortT* thePort
);
```

Parameters

thePort – [in] handle of the port returned from portNew.

Return Value

None

Remarks

None.

3.2 IPMI Layer

These commands take their input arguments and call the port layer to communicate with the CMC. Then they return the bytes that they receive from the CMC and return them in the **resp** argument. For a more detailed description of what these commands do, please consult Chapter 2.

3.2.1 smcIpmiCmdInitialize()

This function initializes the IPMI API layer. This must be the first IPMI function that is called by the application.

Prototype

```
ReturnT smcIpmiCmdInitialize (
     PortT* thePort
);
```

Parameters

ThePort – [in] pointer to the PortT handle through which communication is to take place.

Return Value

IPMI_SUCCESS is returned if the initialization succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

None.

3.2.2 smcIpmiCmdCallbackInit()

This function tells the IPMI layer which function to call if it encounters a communication error. The argument function will be called when the port layer returns a communication error to the IPMI layer.

Prototype

```
ReturnT smcIpmiCmdCallbackInit (
     void (*func)(void* arg),
     void*  arg
);
```

Parameters

func – [in] pointer to the function to be called when the IPMI layer detects a communication error.

arg – [in] argument that will be passed to *func*.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

None.

3.2.3 IpmiGetDeviceId()

This function issues an IMPI Get Device Id command to the CMC.

Prototype

```
ReturnT IpmiGetDeviceId (
     RespT resp
);
```

Parameters

resp – [out] receives the result response byte array from the SMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.4 IpmiGetSELInfo()

This function issues an IMPI Get SEL Info command to the SMC.

Prototype

```
ReturnT IpmiGetSELInfo (
     RespT resp
);
```

Parameters

resp – [out] receives the result response byte array from the SMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.5 IpmiReserveSEL()

This function issues an IMPI Reserve SEL command to the SMC.

Prototype

```
ReturnT IpmiReserveSEL (
     RespT resp
);
```

Parameters

resp – [out] receives the result response byte array from the SMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.6 IpmiGetSELEntry()

This function issues an IMPI Get SEL Entry command to the CMC.

Prototype

```
ReturnT IpmiGetSELEntry (
     int   resID,
     int   recID,
     int   offset,
     int   count,
     RespT resp
);
```

Parameters

```
resID  –  [in] SEL reservation ID
recID  –  [in] SEL record ID
offset –  [in] offset in SEL record to
               retrieve
count  –  [in] number of bytes in SEL
               record to retrieve
resp   – [out] receives the result re-
               sponse byte array from
               the CMC.
```

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.7 IpmiClearSEL()

This function issues an IMPI Get SEL Entry command to the CMC.

Prototype

```
ReturnT IpmiClearSEL (
     int   resID,
     RespT resp
);
```

Parameters

resID – [in] SEL reservation ID returned from Reserve SEL

resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.8 IpmiGetSELTime()

This function issues an IMPI Get SEL Time command to the CMC.

Prototype

```
ReturnT IpmiGetSELTime (
     RespT resp
);
```

Parameters

resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.9 IpmiSetSELTime()

This function issues an IMPI Set SEL Time command to the CMC.

Prototype

```
ReturnT IpmiSetSELTime (
     long  time,
     RespT resp
);
```

Parameters

time – [in] New time to set in the SEL.
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.10 IpmiDeleteSELEntry()

This function issues an IMPI Delete SEL Entry command to the CMC.

Prototype

```
ReturnT IpmiDeleteSELEntry (
     int   resID,
     int   recID,
     RespT resp
);
```

Parameters

resID – [in] SEL reservation ID
recID – [in] SEL record to delete
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.11 IpmiGetSDRRepositoryInfo()

This function issues an IMPI Delete SEL Entry command to the CMC.

Prototype

```
ReturnT IpmiGetSDRRepositoryInfo (
     RespT resp
);
```

Parameters

resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.12 IpmiGetSDR()

This function issues an IMPI Get SDR command to the CMC.

Prototype

```
ReturnT IpmiGetSDR (
     int   resID,
     int   recID,
     int   offset,
     int   count,
     RespT resp
);
```

Parameters

```
resID  –  [in] SDR reservation ID
recID  –  [in] SDR record ID
offset –  [in] offset in SDR record to
               retrieve
count  –  [in] number of bytes in SDR
               record to retrieve
resp   – [out] receives the result re-
               sponse byte array from
               the CMC.
```

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.13 IpmiPartialAddSdr()

This function issues an IMPI Get SDR command to the CMC.

Prototype

```
ReturnT IpmiPartialAddSdr (
     int   resID,
     int   recID,
     int   offset,
     int   last,
     const unsigned char* data,
     int   dataSize,
     RespT resp
);
```

Parameters

resID – [in] SDR reservation ID returned from Reserve SDR Repository

recID – [in] SDR record ID

offset – [in] offset in SDR record to retrieve

last – [in] signifies whether or not this is the last IpmiPartialAddSdr command for this SDR.

data – [in] the new SDR

dataSize – [in] number of bytes in the new SDR

resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.14 IpmiDeleteSDR()

This function issues an IMPI Delete SDR command to the CMC.

Prototype

```
ReturnT IpmiDeleteSDR (
     int   resID,
     int   recID,
     RespT resp
);
```

Parameters

resID – [in] SDR reservation ID
recID – [in] SDR record ID
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.15 IpmiGetSensorReadingFactors()

This function issues an IMPI Get Sensor Reading Factors command to the CMC.

Prototype

```
ReturnT IpmiGetSensorReadingFactors (
     int   sensorID,
     int   reading,
     RespT resp
);
```

Parameters

sensorID – [in] sensor ID
reading – [in] sensor reading to get factors for
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.16 IpmiGetSensorThresholds()

This function issues an IMPI Get Sensor Thresholds command to the CMC.

Prototype

```
ReturnT IpmiGetSensorThresholds (
     int   sensorID,
     RespT resp
);
```

Parameters

sensorID – [in] sensor ID
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.17 IpmiSetSensorThresholds()

This function issues an IMPI Set Sensor Thresholds command to the CMC.

Prototype

```
ReturnT IpmiSetSensorThresholds (
     int   sensorID,
     int   thresholds,
     unsigned char* thresholdData,
     int   count,
     RespT resp
);
```

Parameters

sensorID	–	[in] sensor ID
thresholds	–	[in] bit mask of thresholds to set
thresholdData	–	[in] list of values to set in thresholds
count	–	[in] number of bytes in thresholdData list
resp	–	[out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.18 IpmiGetSensorReading()

This function issues an IMPI Get Sensor Reading command to the CMC.

Prototype

```
ReturnT IpmiGetSensorReading (
     int   sensorID,
     RespT resp
);
```

Parameters

sensorID – [in] Sensor ID
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.19 IpmiSetSensorEventEnable()

This function issues an IMPI Set Sensor Event Enable command to the CMC.

Prototype

```
ReturnT IpmiSetSensorEventEnable (
     int   sensorID,
     int   status,
     int   amask,
     int   dmask,
     RespT resp
);
```

Parameters

sensorID – [in] Sensor ID
status – [in] status byte
amask – [in] assertion mask
dmask – [in] deassertion mask
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.20 IpmiGetSensorEventEnable()

This function issues an IMPI Get Sensor Event Enable command to the CMC.

Prototype

```
ReturnT IpmiGetSensorEventEnable (
     int   sensorID,
     RespT resp
);
```

Parameters

sensorID – [in] Sensor ID
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.21 IpmiSetAlarms()

This function issues an IMPI Set Alarms command to the CMC.

Prototype

```
ReturnT IpmiGetSELEntry (
     int   sensorID,
     RespT resp
);
```

Parameters

sensorID – [in] Sensor ID
resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

3.2.22 IpmiGetAlarms()

This function issues an IMPI Get Alarms command to the CMC.

Prototype

```
ReturnT IpmiGetAlarms (
     RespT resp
);
```

Parameters

resp – [out] receives the result response byte array from the CMC.

Return Value

IPMI_SUCCESS is returned if the call succeeded, otherwise an IPMI error code is returned. Please see the list of error codes in this document.

Remarks

Please see Chapter 2 for more detailed description of the arguments to this command.

This page was intentionally left blank

4 SNMP Agent MIB Guide

4.1 Overview

The I-Bus CMC SNMP Agent (”Agent”) provides an SNMP Management station (”Manager”) with information and control of the I-Bus Chassis Management Controller (”CMC”) through the SNMP Protocol. The protocol is based on the concept of objects that have values. The values of some objects can be changed by the Manager while the values of other objects can only be read. Objects can also be organized into tables. The objects provided by the Agent are described in machine readable format, in a document called ”MIB” (”Management Information Base”). The rest of this guide explains each of the objects provided by the Agent. This guide does not provide a mapping between numbers and meanings. For example, 8 might mean ”no-alarms,” but this guide will only use the term ”no-alarms”. Please review the MIB for the map between numbers and meanings.

4.1.1 Terms Used In This Document

R/W = Read/Write. This means the Manger can change the value of the object.

RO = Read-only. This means that only the agent can change the value of the object.

4.2 sysDescription - R/W.

Typically, the Manager uses this object to record a description of the system.

4.3 productName - RO.

This is always ”I-Bus Chassis Management Controller.”

4.4 sysContact - R/W.

Typically, the Manager uses this object to identify the party responsible for the system.

4.5 sysName - RO.

This is the hostname of the system running the Agent.

4.6 sysLocation - R/W.

Typically, the Manager uses this object to record the location of the system running the Agent.

4.7 sysCommunicationAge - RO.

This indicates the number of seconds that have passed since the last serial communication between the system running the Agent and the CMC itself. A value greater than two indicates that something is not working correctly because the Agent requests information from the CMC every two seconds.

4.8 chassisModel - RO.

This indicates the model number of the chassis in which the CMC is deployed. The CMC retrieves this value from a non-volatile memory store built into the chassis, and passes it on to the Agent when necessary.

4.9 sensorTable - RO.

This table contains a row for each sensor provided by the CMC. Note that the CMC provides more sensors than might be implemented in a particular chassis, but this table will still hold rows for those unimplemented sensors. The following two object descriptions apply to the fields in each row of this table.

4.9.1 sensorIndex - RO.

This identifies the sensor described by this row.

4.9.2 reading - RO.

This indicates the current reading of the sensor.

4.10 thresholdTable - R/W and RO, depending on the field in question (see below).

Each sensor has six thresholds, three above its nominal value, and three below its nominal value. This table has a row for each threshold of each sensor. The following five object descriptions apply to the fields in each row of this table.

4.10.1 thresholdIndex - RO.

This indicates the threshold (whether the threshold is an upper or lower threshold, and its severity) described in this row.

4.10.2 tSensor - RO.

This identifies the sensor described by this row.

4.10.3 thresholdValue - R/W.

This indicates the sensor reading at which this threshold is violated.

4.10.4 tIsDefault - R/W.

This indicates whether or not the values for thresholdValue and tIsEvent will be the same for this row after the CMC is reset or loses power. If either setting is not the default setting, this will be ”no.” Setting this to ”yes” will cause the Agent to instruct the CMC to write the current values of thresholdValue and tIsEvent to non-volatile memory on the CMC board. Setting it to ”no” has no effect.

4.10.5 tIsEvent - R/W.

This indicates whether or not a violation of this threshold will produce an event and an alarm. The Manager can be used to change it to ”yes” or ”no.”

4.11 activeAlarms - R/W.

This indicates which of the three alarms are active. The value will contain ”critical,” ”major,” and ”minor,” if any of those alarms are active. The set of numeric values to which these descriptions map act as a bit field in which bit 3 is always on, bit 2 represents the critical alarm, bit 1 represents the major alarm, and bit 0 represents the minor alarm. The intent it to provide a programmer with convenient access to the alarm states by querying only one object. The Manager can set this to manipulate alarm states.

4.12 critical - R/W.

This indicates if the critical alarm is on. The Manager can control the critical alarm by setting it.

4.13 major - R/W.

This indicates if the major alarm is on. The Manager can control the major alarm by setting it.

4.14 minor - R/W.

This indicates if the minor alarm is on. The Manager can control the minor alarm by setting it.

4.15 alarmCauseTable - RO.

This table contains a row for each sensor that is causing an alarm. Once a sensor reading violates one of the thresholds for which tIsEvent is true, this table will contain a row for that violation. The row will stay in the table until sensor reading returns to normal. This includes hysteresis as explained in the CMC Programmer's Guide. If no Manager has requested the (entire) row, it will remain in the table even after the sensor returns to a normal reading. This ensures that transient alarm conditions are reported to the Manager even if it does not query for them before they go away. As the reading of a sensor moves away from the nominal value, it usually causes a minor alarm first, and then, as it moves further, it causes a major alarm, and then a critical alarm. In this situation, there will be only one row for the sensor, and that row will represent the most severe alarm. The following 6 object descriptions apply to the fields in each row of this table. (violationIndex is not described because it is inaccessible.)

4.15.1 vAlarm - RO.

This indicates the severity of the alarm caused by the threshold violation.

4.15.2 vSensor - RO.

This identifies the sensor that caused the alarm.

4.15.3 vThresholdNum - RO.

This identifies the threshold (whether the threshold is an upper or lower threshold, and its severity) that was violated.

4.15.4 vCurrentReading - RO.

This indicates the current reading of the sensor.

4.15.5 vThresholdValue - RO.

This indicates the value of the threshold that was violated.

4.15.6 vTime - RO.

This indicates the time at which the sensor reading reached the threshold. If the reading goes on to reach the next threshold beyond this one, this row will be "shadowed" (it will no longer appear in the table) by the more severe alarm associated with that threshold. When the reading comes back across that threshold, the row for the more severe alarm will be removed (assuming it has been viewed by a Manager), and this row will reappear. vTime for this row will still indicate the time at which the sensor reading first violated the threshold, rather than the time at which the reading came back across the next threshold.

The following is a sample program that demonstrates how to use the IPMI API. The program can be compiled using the following command:

```
cc –o ipmiapitest ipmiapitest.c –libusipmi
```

```
/*
 * Copyright 2001 I-Bus
 */
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <smc_ipmi_cmd.h>


int   cmdArgc;
int*  cmdArgv;
char* cmdName;

ReturnT getDeviceId(RespT resp);
ReturnT getSELInfo(RespT resp);
ReturnT reserveSEL(RespT resp);
```

```
ReturnT getSELEntry(RespT resp);
ReturnT clearSEL(RespT resp);
ReturnT getSELTime(RespT resp);
ReturnT setSELTime(RespT resp);
ReturnT deleteSELEntry(RespT resp);
ReturnT getSDRRepositoryInfo(RespT resp);
ReturnT reserveSDRRepository(RespT resp);
ReturnT getSDR(RespT resp);
ReturnT partialAddSdr(RespT resp);
ReturnT deleteSDR(RespT resp);
ReturnT getSensorReadingFactors(RespT resp);
ReturnT getSensorThresholds(RespT resp);
ReturnT setSensorThresholds(RespT resp);
ReturnT getSensorReading(RespT resp);
ReturnT setSensorEventEnable(RespT resp);
ReturnT getSensorEventEnable(RespT resp);
ReturnT setSensorAlarms(RespT resp);
ReturnT getSensorAlarms(RespT resp);
ReturnT setAlarms(RespT resp);
ReturnT getAlarms(RespT resp);


typedef struct Cmd {
    char*     name;
    int       argCount;
    ReturnT (*func)(RespT resp);
    char*     argList;
```

```
} CmdT;

CmdT cmds[] = {
    {"getDeviceId", 0, getDeviceId, ""},
    {"getSELInfo", 0, getSELInfo, ""},
    {"reserveSEL", 0, reserveSEL, ""},
    {"getSELEntry", 4, getSELEntry, "resID recID offset count"},
    {"clearSEL", 1, clearSEL, "resId"},
    {"getSELTime", 0, getSELTime, ""},
    {"setSELTime", 1, setSELTime, "time"},
    {"deleteSELEntry", 2, deleteSELEntry, "resID recID"},
    {"getSDRRepositoryInfo", 0, getSDRRepositoryInfo, ""},
    {"reserveSDRRepository", 0, reserveSDRRepository, ""},
    {"getSDR", 4, getSDR, "resID recID offset count"},
    {"partialAddSdr", -5, partialAddSdr, "resID recID offset last data ..."},
    {"deleteSDR", 2, deleteSDR, "resID recID"},
    {"getSensorReadingFactors", 2, getSensorReadingFactors, "sensor reading"},
    {"getSensorThresholds", 1, getSensorThresholds, "sensor"},
    {"setSensorThresholds", -3, setSensorThresholds, "sensor thresholds data ..."},
    {"getSensorReading", 1, getSensorReading, "sensor"},
    {"setSensorEventEnable", 4, setSensorEventEnable, "sensor status amask dmask"},
    {"getSensorEventEnable", 1, getSensorEventEnable, "sensor"},
    {"setSensorAlarms", 2, setSensorAlarms, "sensor mask"},
    {"getSensorAlarms", 1, getSensorAlarms, "sensor"},
    {"setAlarms", 1, setAlarms, "alarm-mask"},
    {"getAlarms", 0, getAlarms, ""},
    {NULL, NULL}
```

```
};


/*
 * Useful prototypes
 */
void usage();
int  init(int argc, char* argv[]);


/*
 * These are set through the side effects of ”init”
 */
char*   myName       = ”ipmiapitest”;
char*   port         = ”/dev/ttyb”;
int     burstDelay;
int     doBurstDelay;
int     burstCount;
int     loopCount = 1;
int     doBurstCount;
int     cmdLen;
unsigned char writebuf[128];
unsigned char readbuf[128];


int
main(int argc, char** argv)
{
```

```
PortT* pPort  = NULL;
int    rVal   = 1;
int    myRval = 1;

myName = argv[0];

rVal = init(argc, argv);

if (0 == rVal)
{
    pPort = portNew(port);

    if (NULL != pPort)
    {
        CmdT* pCmd   = NULL;
        CmdT* theCmd = NULL;

        rVal = smcIpmiCmdInitialize(pPort);

        if (IPMI_SUCCESS == rVal)
        {
            for (pCmd = cmds; NULL == theCmd && NULL != pCmd->name;
                  ++pCmd)
            {
                if (0 == strcasecmp(cmdName, pCmd->name))
                {
                    theCmd = pCmd;
                }
```

```
}

if (NULL != theCmd)
{
    RespT resp = RESPINIT;

    if (((0 <= theCmd->argCount) &&
                (cmdArgc == theCmd->argCount)) ||
        ((0 >  theCmd->argCount) &&
                (cmdArgc >= (-theCmd->argCount))))
    {
        int i = 0;

        for (i = 0; i < loopCount; ++i)
        {
            rVal = (*(theCmd->func))(resp);

            if (IPMI_SUCCESS == rVal)
            {
                int cnt = resp[0];
                unsigned char* xp = resp;

                while (0 < cnt--)
                {
                    printf("%d,", *xp++);
                }
                printf("%d\n", *xp);
            }
```

```
                        else
                        {
                            printf("command failed\n");
                            break;
                        }
                    }
                }
                else
                {
                    printf("Usage: %s %s %s\n", myName,
                        theCmd->name, theCmd->argList);
                }
            }
            else
            {
                printf("%s is not a valid command\n", cmdName);
            }
        }

        myRval = 0;

        portDelete(pPort);
    }
    else
    {
        printf("Cannot open the port!\n");
    }
```

```
    }

    return myRval;
}


void
usage()
{
    (void)fprintf(
        stderr,
        "Usage: %s [-l loop count] [-c burst count ] "
        "[-d burst delay] [-p <port filename>] command-name"
        " command-args\n", myName);
}


/*
 * ipmiapitest [ -d <burst delay> ] [ -c <burst count> ]
 *             [ -p <serial port> ] <command bytes>
 *
 */
int
init(int argc, char* argv[])
{
    extern char* optarg;
    extern int   optind;
    int          rVal      = 1;
```

```
int          c         = 0;
int          errflg    = 0;

/*
 * Handle the arguments
 */
while ((c = getopt(argc, argv, "c:d:l:p:")) != EOF)
{
    switch (c) {
    case 'c':
        burstCount = strtol(optarg, NULL, 0);
        doBurstCount++;
        //(void)printf("count = %d\n", burstCount);
        break;

    case 'd':
        burstDelay = strtol(optarg, NULL, 0);
        doBurstDelay++;
        //(void)printf("delay = %d\n", burstDelay);
        break;

    case 'l':
        loopCount = strtol(optarg, NULL, 0);
        //(void)printf("loopcnt = %d\n", loopCount);
        break;

    case 'p':
        port = optarg;
```

```
            //(void)printf(”portname = %s\n”, port);
            break;

        case ’?’:
            errflg++;
            break;
        }
    }

    if (!errflg)
    {
        cmdName = argv[optind];
        optind++;
        cmdArgc = argc – optind;

        //printf(”remaining arg count = %d\n”, argsLeft);

        if (0 <= cmdArgc)
        {
            cmdArgv = (int *)calloc(cmdArgc, sizeof (int));

            if (NULL != cmdArgv)
            {
                int* pi = NULL;

                /* Gather the args into a command string */
                for (pi = cmdArgv; optind < argc; optind++)
                {
```

```
                    *pi++ = strtol(argv[optind], NULL, 0);
                }

                rVal = 0;
            }
            else
            {
                rVal = 2;
            }
        }
        else
        {
            usage();
            rVal = 4;
        }
    }
    else
    {
        usage();

        rVal = 1;
    }

    return rVal;
}
```

```
ReturnT getDeviceId(RespT resp)
{
    return IpmiGetDeviceId(resp);
}


ReturnT getSELInfo(RespT resp)
{
    return IpmiGetSELInfo(resp);
}


ReturnT reserveSEL(RespT resp)
{
    return IpmiReserveSEL(resp);
}


ReturnT getSELEntry(RespT resp)
{
    return IpmiGetSELEntry(cmdArgv[0], cmdArgv[1], cmdArgv[2],
           cmdArgv[3], resp);
}


ReturnT clearSEL(RespT resp)
```

```
{
    return IpmiClearSEL(cmdArgv[0], resp);
}


ReturnT getSELTime(RespT resp)
{
    return IpmiGetSELTime(resp);
}


ReturnT setSELTime(RespT resp)
{
    return IpmiSetSELTime(cmdArgv[0], resp);
}


ReturnT deleteSELEntry(RespT resp)
{
    return IpmiDeleteSELEntry(cmdArgv[0], cmdArgv[1], resp);
}


ReturnT getSDRRepositoryInfo(RespT resp)
{
    return IpmiGetSDRRepositoryInfo(resp);
}
```

```
ReturnT reserveSDRRepository(RespT resp)
{
    return IpmiReserveSDRRepository(resp);
}


ReturnT getSDR(RespT resp)
{
    return IpmiGetSDR(cmdArgv[0], cmdArgv[1], cmdArgv[2], cmdArgv[3],
            resp);
}


ReturnT partialAddSdr(RespT resp)
{
    ReturnT        rVal      = IPMI_ERR_BADARG;
    unsigned char* data      = NULL;
    int            byteCount = cmdArgc – 4;

    data = (unsigned char*) calloc(byteCount, sizeof (unsigned char));

    if (NULL != data)
    {
        int i = 0;

        /* Transfer the remaining args to the data array */
```

```
        for (i = 0; i < byteCount; ++i)
        {
            data[i] = cmdArgv[i + 4];
        }

        rVal = IpmiPartialAddSdr(
            cmdArgv[0],
            cmdArgv[1],
            cmdArgv[2],
            cmdArgv[3],
            data,
            byteCount,
            resp
        );
    }
    else
    {
        rVal = IPMI_ERR_NOMEM;
    }

    return rVal;
}


ReturnT deleteSDR(RespT resp)
{
    return IpmiDeleteSDR(cmdArgv[0], cmdArgv[1], resp);
```

```
}


ReturnT getSensorReadingFactors(RespT resp)
{
    return IpmiGetSensorReadingFactors(cmdArgv[0], cmdArgv[1], resp);
}


ReturnT getSensorThresholds(RespT resp)
{
    return IpmiGetSensorThresholds(cmdArgv[0], resp);
}


ReturnT setSensorThresholds(RespT resp)
{
    ReturnT        rVal      = IPMI_FAIL;
    unsigned char* data      = NULL;
    int            byteCount = cmdArgc – 2;

    data = (unsigned char*) calloc(byteCount, sizeof (unsigned char));

    if (NULL != data)
    {
        int i = 0;

        /* Transfer the remaining args to the data array */
```

```
        for (i = 0; i < byteCount; ++i)
        {
            data[i] = cmdArgv[i + 2];
        }

        rVal = IpmiSetSensorThresholds(cmdArgv[0], cmdArgv[1],
                  data, byteCount, resp);
    }
    else
    {
        rVal = IPMI_ERR_NOMEM;
    }

    return rVal;
}


ReturnT getSensorReading(RespT resp)
{
    return IpmiGetSensorReading(cmdArgv[0], resp);
}


ReturnT setSensorEventEnable(RespT resp)
{
```

```
    return IpmiSetSensorEventEnable(
            cmdArgv[0],
            cmdArgv[1],
            cmdArgv[2],
            cmdArgv[3],
            resp);
}


ReturnT getSensorEventEnable(RespT resp)
{
    return IpmiGetSensorEventEnable(cmdArgv[0], resp);
}


ReturnT setSensorAlarms(RespT resp)
{
    return IpmiSetSensorAlarms(cmdArgv[0], cmdArgv[1], resp);
}


ReturnT getSensorAlarms(RespT resp)
{
    return IpmiGetSensorAlarms(cmdArgv[0], resp);
}


ReturnT setAlarms(RespT resp)
```

```
{
    return IpmiSetAlarms(cmdArgv[0], resp);
}


ReturnT getAlarms(RespT resp)
{
    return IpmiGetAlarms(resp);
}
```

This page was intentionally left blank

B

backplane: A device inside the chassis that contains slots, or sockets, for plugging in cards or cables.

bidirectional parallel port: An eight–bit port that can be used for an input as well as an output device.

bus: One or more electrical conductors that transmit power or binary data to the various sections of a computer or any common pathway between hardware devices. A computer bus connects the CPU to its main memory and the memory banks that reside on the control units of the peripheral devices. It is made up of two parts. Addresses are sent over the address bus to signal a memory location, and the data are transferred over the data bus to that location.

C

card cage: A cabinet or metal frame that holds printed circuit cards.

CMOS (Complementary Metal Oxide Semiconductor): A technique of arranging transistors which uses very low power.

CompactPCI: A ruggedized variation of the PCI bus using a high density 2 mm contact spacing pin and socket connector for interface between the cards and a passive backplane.

D

disk access LED: The LED located on the front control panel that indicates when the hard disk drive is active.

DRAM (Dynamic Random Access Memory): The main memory in your computer. It needs to be refreshed by a memory controller or it loses its information.

drive bay: Area in the chassis where drives are mounted.

E

electrostatic discharge (ESD): Stationary electrical charges in which no current flows. ESD can be prevented by wearing a wrist strap attached to a ground post on a static mat.

EMI (ElectroMagnetic Interference): Noise generated by the switching action of the power supply and other system components. Conducted EMI is interference generally conducted into the power line, and is normally controlled with a line filter. Radiated EMI is that portion that radiates into free space. One way to suppress it is by enclosing circuitry in a metal case.

EPROM (Erasable Programmable Read Only Memory): A programmable device which stores information regardless of power.

expansion card: A printed circuit board that plugs into an expansion slot.

F

floppy drive: A device for reading the information contained on external, portable computer disks called floppy disks.

front control panel: The small panel on the front of the computer that contains the power switch, reset switch, Power ON LED, the disk access LED, and the keyboard connector.

H

hard drive: A data storage device. Hard drives magnetically store computer data on spinning internal disks.

hold–down bar: A metal bar located in the I/O bay of the chassis. It is used to keep I/O cards firmly seated in their slots. (There is no hold–down bar in CompactPCI systems.)

Hot Swap: The process of replacing a failed component while the rest of the system continues working and functioning normally.

I

IDE (Integrated Drive Electronics): A standard of signalling and communicating with a device.

I/O card: A printed circuit board that plugs into an I/O slot.

I/O slot: A slot for plugging in additional I/O cards to expand the capability of a computer.

ISA: The original IBM/PC clone plug–in board standard.

K

keyboard connector: The five–pin connector located on the front control panel.

kilobyte (KB): 1,024 bytes.

L

LED: Light Emitting Diode. Long–lasting light emitters usually used as indicators.

load board: A board having specific resistance to current flow.

P

parallel port: I/O connector used to hook up a printer or other parallel interface device. The parallel port is usually a 25–pin female DB25 connector.

PCI(Peripheral Component Interconnect): An optional slot standard for plug–in boards

port: Ports are used to connect peripheral devices such as external drives and printers to your computer.

power good: Signal used to prevent the computer from starting until the power has stabilized. The power good line switches from 0 to +5 volts within one tenth to one half second after the power supply reaches normal voltage levels. Whenever low input voltage causes the output voltage to fall below operating levels, the power good signal goes back to zero.

power ON/diagnostic LED: The LED located on the front control panel that indicates that power is present in the computer.

power supply: Electrical system that converts AC current from the wall outlet into the DC currents required by the computer circuitry. In a personal computer, +5, –5, +12 and –12 voltages are generated.

power switch: Located on the front control panel, the power switch turns power ON to the computer.

R

RAID (Redundant Array of Inexpensive Disks): A storage technology using an array of two or more disks to redundantly store information. If one disk fails in a RAID array, the unit continues to function without loss of data.

RAM (Random Access Memory): The memory used to execute applications while your computer is turned ON. When you turn your computer OFF, all data stored in RAM is lost.

real–time clock (RTC): A periodic interrupt used to derive local time.

reset switch: Button or key that reboots the computer. All current activities are stopped cold and any data in memory are lost.

retaining bracket: The bracket on the back of the chassis that holds connectors from the board, usually a DB9 for serial port, a DB25 for parallel port, and mini–DIN connectors for keyboard and mouse.

S

SCSI (Small Computer System Interface): A high speed, general purpose interface to storage devices.

serial port: A two–channel port, one channel used for ”In” transmissions and one for ”Out” transmissions.

SDR (Sensor Data Record): A record of data about a sensor. The full description of the contents of these records can be found in IPMI, verion 1.0, section 28.1.

SEL (System Event Log): A non–volatile storage area and associated interfaces for storing system platform event information for later retrieval.

W

watchdog timer: A device that watches for CPU inactivity and then resets the CPU after a specified duration of inactivity.

LIMITED WARRANTY

I-Bus warrants this product to be free of defects in material and workmanship for an initial period of one (1) year from date of delivery to the original purchaser from I-Bus.

During this period, I-Bus will, at its option, repair or replace this product at no additional charge to the purchaser, except as set forth in this warranty agreement.

I-Bus will, at its option, repair or replace this product at no additional charge to the purchaser, if the defect is related to the I-Bus manufactured product, such as power supply, backplanes, other chassis components, or CPUs. I-Bus is not liable for any defects in material or workmanship of any peripherals, products or parts which I-Bus does not design or manufacture. However, I-Bus will honor the original manufacturer's warranty for these products.

I-Bus will analyze the defective component and the customer will be charged.

Receipt of damaged goods voids the I-Bus warranty.

Repair parts and replacement products will be furnished on an exchange basis and will be either new or reconditioned. All replacement parts and products shall become the property of I-Bus, if such parts or products are provided under this warranty agreement. In the event a defect is not related to the I-Bus manufactured product, I-Bus shall repair or replace the defective parts at purchaser's cost and deliver the defective parts to the purchaser.

This Limited Warranty shall not apply if the product has been misused, carelessly handled, defaced, modified or altered, or if unauthorized repairs have been attempted by others.

The above warranty is the only warranty authorized by I-Bus and is in lieu of any implied warranties, including implied warranty of merchantability and fitness for a particular purpose.

In no event will I-Bus be liable for any such damage as lost business, lost profits, lost savings, downtime or delay, labor, repair or material cost, injury to person or property or any similar or dissimilar consequential loss or damage incurred by purchaser, even if I-Bus has been advised of the possibility of such losses or damages.

In order to obtain warranty service, the product must be delivered to the I-Bus facility, or to an authorized I-Bus service representative, with all included parts and accessories as originally shipped, along with proof of purchase and a Returned Merchandise Authorization (RMA) number.

The RMA number is obtained, in advance, from I-Bus Customer Service Department and is valid for 30 days. The RMA number must be clearly marked on the exterior of the original shipping container or equivalent. Purchaser will be responsible and liable for any missing or damaged parts. Purchaser agrees to pay shipping charges one way, and to either insure the product or assume the liability for loss or damage during transit. Ship to:

I-Bus (see page 2 to get I-BUS address)

ATTENTION: RMA REPAIR DEPT.

RMA ####

Appendix 3 – Limited Warranty

This page was intentionally left blank