- IPMI -

Intelligent Platform Management Interface Specification

v1.5

Document Revision 1.1
February 20, 2002

Intel Hewlett-Packard NEC Dell

Revision History

Date	Ver	Rev	Modifications
9/16/98	1.0	1.0	IPMI v1.0 Initial release
8/26/99	1.0	1.1	Errata Revision. Incorporated errata from revision 1 or the Errata and Clarifications for the IPMI v1.0 specification.
2/21/01	1.5	1.0	IPMI v1.5 Initial release
2/20/02	1.5	1.1	Updated to include addenda and errata

IPMI NON-DISCLOSURE AGREEMENT

DO NOT download these files (collectively, the "Confidential Information") until you have carefully read the following terms and conditions. By downloading the Confidential Information you agree to the terms of this Agreement. If you do not wish to so agree, do not download the Confidential Information.

1. Confidential Information. The confidential, proprietary and trade secret information being disclosed ("Confidential Information"), is that information marked with a "confidential", "proprietary", or similar legend, and is described as:

   Confidential Information: Intelligent Platform Management Interface Specification (v1.5), Intelligent Platform Management Bus Bridge Specification (v1.0), Intelligent Chassis Management Bus Bridge Specification (v1.0)
   CONFIDENTIAL INFORMATION IS PROVIDED SOLELY FOR YOUR INTERNAL EVALUATION AND REVIEW TO DETERMINE WHETHER TO ADOPT THE SPECIFICATIONS BY SIGNING A SEPARATE ADOPTER'S AGREEMENT. THE RECEIVING PARTY IS NOT LICENSED TO IMPLEMENT THE SPECIFICATIONS UNLESS OR UNTIL AN ADOPTER'S AGREEMENT IS EXECUTED.

   Disclosing party's representatives for disclosing Confidential Information is: Fadi Zuhayri (fadi.zuhayri@intel.com)

2. Obligations of Receiving Party. The receiving party will maintain the confidentiality of the Confidential Information of the disclosing party with at least the same degree of care that it uses to protect its own confidential and proprietary information, but no less than a reasonable degree of care under the circumstances. The receiving party will not disclose any of the disclosing party's Confidential Information to employees or to any third parties except to the receiving party's employees, parent company and majority-owned subsidiaries who have a need to know and who agree to abide by nondisclosure terms at least as comprehensive as those set forth herein; provided that the receiving party will be liable for breach by any such entity. The receiving party will not make any copies of Confidential Information received from the disclosing party except as necessary for its employees, parent company and majority-owned subsidiaries with a need to know. Any copies which are made will be identified as belonging to the disclosing party and marked "confidential", "proprietary" or with a similar legend.

3. Period of Non-Assertion. Unless a shorter period is indicated below, the disclosing party will not assert any claims for breach of this Agreement or misappropriation of trade secrets against the receiving party arising out of the receiving party's disclosure of disclosing party's Confidential Information made more than five (5) years from the date of receipt of the Confidential Information by the receiving party. However, unless at least one of the exceptions set forth in Section 4 below has occurred, the receiving party will continue to treat such Confidential Information as the confidential information of the disclosing party and only disclose any such Confidential Information to third parties under the terms of a non-disclosure agreement.

4. Termination of Obligation of Confidentiality. The receiving party will not be liable for the disclosure of any Confidential Information which is: (a) rightfully in the public domain other than by a breach of this Agreement of a duty to the disclosing party; (b) rightfully received from a third party without any obligation of confidentiality; (c) rightfully known to the receiving party without any limitation on use or disclosure prior to its receipt from the disclosing party; (d) independently developed by employees of the receiving party; or (e) generally made available to third parties by the disclosing party without restriction on disclosure.

5. Title. Title or the right to possess Confidential Information as between the parties will remain in the disclosing party.

6. No Obligation of Disclosure; Termination The disclosing party may terminate this Agreement at any time without cause upon written notice to the other party; provided that the receiving party's obligations with respect to information disclosed during the term of this Agreement will survive any such termination. The disclosing party may, at any time: (a) cease giving Confidential Information to the other party without any liability, and/or (b) request in writing the return or destruction of all or part of its Confidential Information previously disclosed, and all copies thereof, and the receiving party will promptly comply with such request, and certify in writing its compliance.

7. General.
   (a) This Agreement is neither intended to nor will it be construed as creating a joint venture, partnership or other form of business association between the parties, nor an obligation to buy or sell products using or incorporating the Confidential Information.
   (b) No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon either party in this Agreement or by the transfer of any information by one party to the other party as contemplated hereunder, either expressly, by implication, inducement, estoppel or otherwise, and that any license under any such intellectual property rights must be express and in writing.
   (c) The failure of either party to enforce any right resulting from breach of any provision of this Agreement will not be deemed a waiver of any right relating to a subsequent breach of such provision or of any other right hereunder.
   (d) This Agreement will be governed by the laws of the State of Delaware without reference to conflict of laws principles.
   (e) This Agreement constitutes the entire agreement between the parties with respect to the disclosure(s) of Confidential Information described herein, and may not be amended except in a writing signed by a duly authorized representative of the respective parties. Any other agreements between the parties, including non-disclosure agreements, will not be affected by this Agreement.

Table of Contents

List of Figures

List of Tables

1. Introduction

This document presents the base specifications for the *Intelligent Platform Management Interface* (IPMI) architecture. The IPMI specifications define standardized, abstracted interfaces to the platform management subsystem. IPMI includes the definition of interfaces for extending platform management between board within the main chassis, and between multiple chassis.

The term "platform management" is used to refer to the monitoring and control functions that are built in to the platform hardware and primarily used for the purpose of monitoring the health of the system hardware. This typically includes *monitoring* elements such as system temperatures, voltages, fans, power supplies, bus errors, system physical security, etc. It includes automatic and manually driven *recovery* capabilities such as local or remote system resets and power on/off operations. It includes the *logging* of abnormal or 'out-of-range' conditions for later examination and *alerting* where the platform issues the alert without aid of run-time software. Lastly it includes *inventory* information that can help identify a failed hardware unit.

This document is the main specification for IPMI. It defines the overall architecture, common commands, event formats, data records, and capabilities used across IPMI-based systems and peripheral chassis. This includes the specifications for IPMI management via LAN, serial/modem, PCI Management bus, and the local interface to the platform management. In addition to this document, there is a set of separate supporting specifications:

- The *Intelligent Platform Management Bus* (IPMB) is an I^2C*-based bus that provides a standardized interconnection between different boards within a chassis. The IPMB can also serve as a standardized interface for auxiliary or 'emergency' management add-in cards.
- *IPMB v1.0 Address Allocation* documents the different ranges and assignments of addresses on the IPMB.
- The *Intelligent Chassis Management Bus* (ICMB) provides a standardized interface for platform management information and control between chassis. The ICMB is designed so it can be implemented with a device that connects to the IPMB. This allows the ICMB to be implemented as an add-on to systems that have an existing IPMB. See [ICMB] for more information.
- The *Platform Event Trap Format* specification defines the format of SNMP traps used for alerts.
- The *Platform Management FRU Information Storage Definition* defines the format of Field Replaceable Unit information (information such as serial numbers and part numbers for various replaceable boards and other components) accessible in an IPMI-based system.

The implementation of certain aspects of IPMI may require access to other specifications and documents that are not part Refer to the *Reference Documents* section below, for these and other supporting documents.

1.1 Audience

This document is written for engineers and system integrators involved in the design of and interface to platform management hardware, and System Management Software (SMS) developers. Familiarity with microcontrollers, software programming, and PC and Intel server architecture is assumed. For basic and/or supplemental information, refer to the appropriate reference documents.

1.2 Reference Documents

The following documents are companion and supporting specifications for IPMI and associated interfaces:

[ACPI 1.0] *Advanced Configuration and Power Interface Specification*, Revision 1.0b, February 8, 1999. ©1999, Copyright © 1996, 1997, 1998, 1999 Intel Corporation, Microsoft Corporation, Toshiba Corp. http://www.teleport.com/~acpi/

[ACPI 2.0] *Advanced Configuration and Power Interface Specification*, Revision 2.0, July 27, 2000. ©1996, 1997, 1998, 1999, 2000 Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation. http://www.teleport.com/~acpi/

[ADDR] *IPMB v1.0 Address Allocation, ©2001 Intel Corporation*, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation. This document specifies the allocation of I^2C addresses on the IPMB. http://developer.intel.com/design/servers/ipmi

[ASF] Alert Standard Format v1.0 Specification, ©2001, Distributed Management Task Force. http://www.dmtf.org

[CBCP] *Proposal for Callback Control Protocol (CBCP)*, draft-ietf-pppext-callback-cp-02.txt, N. Gidwani, Microsoft, July 19, 1994. As of this writing, the specification is available via the Microsoft Corporation web site: http://www.microsoft.com

[FRU] *Platform Management FRU Information Storage Definition v1.0,* ©1999 Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation. Provides the field definitions and format of Field Replaceable Unit (FRU) information. http://developer.intel.com/design/servers/ipmi

[I^2C] *The I^2C Bus And How To Use It,* ©1995, Philips Semiconductors. This document provides the timing and electrical specifications for I^2C busses.

[ICMB] *Intelligent Chassis Management Bus Bridge Specification v1.0,* rev. 1.2, © 2000 Intel Corporation. Provides the electrical, transport protocol, and specific command specifications for the ICMB and information on the creation of management controllers that connect to the ICMB. http://developer.intel.com/design/servers/ipmi

[IPMB] *Intelligent Platform Management Bus Communications Protocol Specification v1.0,* ©1998 Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation. This document provides the electrical, transport protocol, and specific command specifications for the IPMB. http://developer.intel.com/design/servers/ipmi

[MSVT] *Windows Platform Design Notes, Building Hardware and Firmware to Complement Microsoft Windows Headless Operation,* ©2001, Microsoft Corporation. http://www.microsoft.com

[PET] *IPMI Platform Event Trap Format Specification v1.0,* ©1998, Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation. This document specifies a common format for SNMP Traps for platform events.

[RFC826] *An Ethernet Address Resolution Protocol -- or -- Converting Network Protocol Addresses to 48-bit Ethernet Address for Transmission on Ethernet Hardware,* David C. Plummer, November 1982

[RFC1319] *RFC 1319, The MD2 Message-Digest Algorithm,* B. Kaliski, RSA Laboratories, April 1992.

[RFC1321] *RFC 1321, The MD5 Message-Digest Algorithm,* R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc. April, 1992.

[RFC1332] *RFC 1332, The PPP Internet Protocol Control Protocol (IPCP),* G. McGreggor, Merit, May 1992.

[RFC1334] *RFC 1334, PPP Authentication Protocols,* B. Lloyd, L&A, W. Simpson, Daydreamer, October 1992. Document includes specification for PAP (Password Authentication Protocol).

[RFC1661] *RFC 1661, STD 51, The Point-to-Point Protocol (PPP),* Simpson, W., Editor, Daydreamer, July 1994.

[RFC1662] *RFC 1662, STD 51, PPP in HDLC-like Framing,* Simpson, W., Editor, Daydreamer, July 1994.

[RFC1994] *RFC 1994, PPP Challenge Handshake Authentication Protocol (CHAP),* Simpson, W., Editor, Daydreamer August 1994.

[RFC2153] *RFC 2153, PPP Vendor Extensions,* Simpson, W., Daydreamer, May 1997.

[RFC 2433] *RFC 2433, Microsoft PPP CHAP Extensions,* G. Zorn / S. Cobb, Microsoft Corporation, October 1998

[RFC 2759] *RFC 2759, Microsoft PPP CHAP Extensions, Version 2,* G. Zorn, Microsoft Corporation, January 2000

[SMBIOS] *System Management BIOS Specification, Version 2.3.1,* © 1997, 1999 American Megatrends Inc., Award Software International, Compaq Computer Corporation, Dell Computer Corporation, Hewlett-Packard Company, Intel Corporation, International Business Machines Corporation, Phoenix Technologies Limited, and SystemSoft Corporation.

[SMBUS] *System Management Bus (SMBus) Specification, Version 2.0,* ©2000, Duracell Inc., Fujitsu Personal Systems Inc., Intel Corporation, Linear Technology Corporation, Maxim Integrated Products, Mitsubishi Electric Corporation, Moltech Power Systems, PowerSmart Inc., Toshiba Battery Co., Ltd., Unitrode Corporation, USAR Systems.

[TAP] *Telocator Access Protocol version 1.8, February 04, 1997*. ©1997, Personal Communications Industry Association. http://www.pcia.com (As of this writing, the document is found under 'Wireless Resource Center | Protocols', or: http://www.pcia.com/wireres/protocol.htm.) This document specifies a protocol for sending an alphanumeric page by connecting to a paging service via a serial modem.

[TIA-602] *TIA/EIA Standard: Data Transmission Systems and Equipment - Serial Asynchronous Automatic Dialing and Control, TIA/EIA 602, June 1992.* © 1992, Telecommunications Industry Association. Also available from the Electronic Industries Association. This document specifies the dialing protocol commonly used in asynchronous serial modems.

[WFM] *Wired for Management Baseline Version 2.0 Release,* ©1998, Intel Corporation. Attachment A, UUIDs and GUIDs, provides information specifying the formatting of the IPMI Device GUID and FRU GUID and the System Management BIOS (SM BIOS) UUID unique IDs.

1.3 Conventions and Terminology

If not explicitly indicated, bits in figures are numbered with the most significant bit on the left and the least significant bit on the right.

This document uses the following terms and abbreviations:

Table 1-1, Glossary

Term	Definition
Asserted	Active-high (positive true) signals are asserted when in the high electrical state (near power potential). Active-low (negative true) signals are asserted when in the low electrical state (near ground potential).
BMC	Baseboard Management Controller
Bridge	The circuitry that connects one computer bus to another, allowing an agent on one to access the other.
Byte	An 8-bit quantity.
CMOS	In terms of this specification, this describes the PC-AT compatible region of battery-backed 128 bytes of memory, which normally resides on the baseboard.
Deasserted	A signal is deasserted when in the inactive state. Active-low signal names have "_L" appended to the end of the signal mnemonic. Active-high signal names have no "_L" suffix. To reduce confusion when referring to active-high and active-low signals, the terms one/zero, high/low, and true/false are not used when describing signal states.
Diagnostic Interrupt	A non-maskable interrupt or signal for generating diagnostic traces and 'core dumps' from the operating system. Typically NMI on IA-32 systems, and an INIT on Itanium™-based systems.
Dword	Double word is a 32-bit (4 byte) quantity.
EEPROM	Electrically Erasable Programmable Read Only Memory.
EvM	Notation for 'Event Message'. See text for definitions of 'Event Message'.
FPC	Front Panel Controller.
FRB	Fault Resilient Booting. A term used to describe system features and algorithms that improve the likelihood of the detection of, and recovery from, processor failures in a multiprocessor system.
FRU	Field Replaceable Unit. A module or component which will typically be replaced in its entirety as part of a field service repair operation.
Hard Reset	A reset event in the system that initializes all components and invalidates caches.
I^2C	Inter-Integrated Circuit bus. A multi-master, 2-wire, serial bus used as the basis for the Intelligent Platform Management Bus.
ICMB	Intelligent Chassis Management Bus. A serial, differential bus designed for IPMI messaging between host and peripheral chassis. Refer to [ICMB] for more information.
IERR	Internal Error. A signal from the Intel Architecture processors indicating an internal error condition.
IPM	Intelligent Platform Management.
IPMB	Intelligent Platform Management Bus. Name for the architecture, protocol, and implementation of a special bus that interconnects the baseboard and chassis electronics and provides a communications media for system platform management information. The bus is built on I^2C and provides a communications path between 'management controllers' such as the BMC, FPC, HSC, PBC, and PSC.
ISA	Industry Standard Architecture. Name for the basic 'PC-AT' functionality of an Intel Architecture computer system.
KB	1024 bytes
LUN	Logical Unit Number. In the context of the Intelligent Platform Management Bus protocol, this is a sub-address that allows messages to be routed to different 'logical units' that reside behind the same I^2C slave address.

NMI	Non-maskable Interrupt. The highest priority interrupt in the system, after SMI. This interrupt has traditionally been used to notify the operating system fatal system hardware error conditions, such as parity errors and unrecoverable bus errors. It is also used as a Diagnostic Interrupt for generating diagnostic traces and 'core dumps' from the operating system.
MD2	RSA Data Security, Inc. MD2 Message-Digest Algorithm. An algorithm for forming a 128-bit digital signature for a set of input data.
MD5	RSA Data Security, Inc. MD5 Message-Digest Algorithm. An algorithm for forming a 128-bit digital signature for a set of input data. Improved over earlier algorithms such as MD2.
Payload	For this specification, the term 'payload' refers to the information bearing fields of a message. This is separate from those fields and elements that are used to transport the message from one point to another, such as address fields, framing bits, checksums, etc. In some instances, a given field may be both a payload field and a transport field.
PEF	Platform Event Filtering. The name of the collection of IPMI interfaces in the IPMI v1.5 specification that define how an IPMI Event can be compared against 'filter table' entries that, when matched, trigger a selectable action such as a system reset, power off, alert, etc.
PERR	Parity Error. A signal on the PCI bus that indicates a parity error on the bus.
PET	Platform Event Trap. A specific format of SNMP Trap used for system management alerting. Used for IPMI Alerting as well as alerts using the ASF specification. The trap format is defined in the PET specification. See [PET] and [ASF] for more information.
POST	Power On Self Test.
Re-arm	Re-arm, in the context of this document, refers to resetting internal device state that tracks that an event has occurred such that the device will re-check the event condition and re-generate the event if the event condition exists.
SDR	Sensor Data Record. A data record that provides platform management sensor type, locations, event generation, and access information.
SEL	System Event Log. A non-volatile storage area and associated interfaces for storing system platform event information for later retrieval.
SERR	System Error. A signal on the PCI bus that indicates a 'fatal' error on the bus.
SMI	System Management Interrupt.
SMIC	Server Management Interface Chip. Name for one type of system interface to an IPMI Baseboard Management Controller.
SMM	System Management Mode. A special mode of Intel IA-32 processors, entered via an SMI. SMI is the highest priority non-maskable interrupt. The handler code for this interrupt is typically located in a physical memory space that is only accessible while in SMM. This memory region is typically loaded with SMI Handler code by the BIOS during POST.
SMS	System Management Software. Designed to run under the OS.
Soft Reset	A reset event in the system which forces CPUs to execute from the boot address, but does not change the state of any caches or peripheral devices.
Word	A 16-bit quantity.

1.4 Background - Architectural Goals

A number of goals/principles influence the design and implementation of a platform management subsystem that works across multiple platforms. The abstracted, modular, extensible interfaces specified in this document seek to satisfy those goals. The following review is provided to give a framework to assist in the evaluation options in the implementation of this specification.

Provide Layered Management Value

- Provide management value at each level of integration, and have the net value increase as each level is added. I.e. progressing from processor, through chip set, BIOS, baseboard, baseboard with management circuitry, with onboard networking, with intelligent controllers, with managed chassis, system management software, with Remote Management Cards, etc.
- Maintain modularity so that one level does not carry undue cost burden for another. Levels should retain value if separated. Avoid burdening baseboard with cost for chassis-specific management

functions. Avoid building in functions that unduly impede OEMs in providing their own chassis management features.

– Drive intelligence to appropriate level. Don't put complexity in at a level if the next higher level can handle it. I.e. don't do something with a microcontroller if the system's host processor can do it more flexibly and economically.

Plan for Evolution and Re-use

– Architect so existing implementations can be cleanly extended with new functionality, without requiring existing functionality to be re-implemented or redesigned.

– Architect for product families. Avoid 'local optimizations' that benefit one product at the cost of future projects.

– Knowledge and understanding of the architecture is also a valuable commodity. "Re-inventing the wheel" often means retraining the wheel user. Design to preserve the knowledge base of developers, testers, salespersons, and customers by maintaining consistency in architecture and implementation - in hardware implementation, firmware, software, protocols, and interfaces.

– Design for the economic incorporation of changes in the population and implementation of baseboard and remote sensors. Architect to minimize the impact to hardware and software when sensor population or sensor hardware interfaces change.

– Design to maximize 'self configurability' in system management software. I.e. 'Plug 'N Play'. Provide platform resident discovery mechanisms, such as standardized tables, discovery mechanisms, etc. to reduce or eliminate the need to 'customize' system software for different platforms.

Provide Scalability

– Architecture should scale from entry through enterprise and data center class server systems. Architecture should be adaptable from single board and single chassis, through multi-board and multi-chassis systems.

– Apply 'Layered Management Value' concept. Low-end solutions should be a proper functional subset of higher end solutions. Entry solutions should not carry undue burden for higher class systems.

Support OEM Extensibility:

– Provide clean points for OEM extension and integration.

– Provide OEM support in protocol and command specifications. Reserve command numbers, sensor numbers, etc. for OEM extension.

1.5 New for IPMI v1.5

IPMI v1.5 is an extension of the v1.0 specification. IPMI v1.5 also includes learnings, feedback, and features gathered from industry review and experiences deploying IPMI v1.0 enabled systems.

The following goals guided the creation of the IPMI v1.5 specification:

- Help enable "Always Available Manageability" by enabling new access media: LAN, Serial/Modem, and PCI Management Bus.
- Extend "Autonomous Manageability" by defining new automatic alerting and recovery mechanisms.
- Synch-up with and support emergent and existing standards such as PPP, the DMTF Pre-OS Working Group 'Alert Standard Forum' specification, SMBus 2.0, Compact PCI, and the PCI Management Bus.
- Retain as much backward compatibility with IPMI v1.0 as feasible

The following presents a brief summary of some of the more significant additions and enhancements in the IPMI v1.5 specification:

Serial/Modem Messaging and Alerting

The IPMI v1.5 specification defines how IPMI messaging can be accomplished via a direct serial or external modem connection to the BMC. It also includes the specifications for generating alerts, numeric pages, and alphanumeric pages on events.

Serial Port Sharing

This is a capability that works in conjunction with serial/modem messaging and alerting and allows the baseboard serial connector to be shared between use by the BMC and use by the system. This enables coordination between system features such as console redirection and allows the serial connection to be used for run-time applications while still allowing it to be remotely accessed for 'emergency' management.

Boot Options

IPMI v1.5 includes a boot options 'mailbox' that can be used to direct the operation of BIOS and the booting process following a system power up or reset operation.

LAN Messaging and Alerting

The specification defines how IPMI messaging can be accomplished via a LAN connection to the BMC. It also includes the specifications for generating PET format SNMP traps on events.

Extended BMC Messaging 'Channel Model'

IPMI v1.0 introduced a limited capability to use a 'Channel Number' capability that was primarily used for routing messages to the IPMB. IPMI v1.5 expands on the use of channel numbers as a general mechanism for routing messages between different media and organizing the access

Additional Sensor and Event Types

Several new sensor types have been added to IPMI v1.5, including a Slot/Connector sensor for monitoring hot-plug slot status, an ACPI System Power State sensor to support out-of-band monitoring of the system power state, and an enhanced Watchdog sensor that supports events generated by the standardized watchdog timer function.

Platform Event Filtering (PEF)

Platform Event Filtering (PEF) provides a mechanism for configuring the BMC to taking selected actions on event messages that it receives or has internally generated. These actions include operations such as system power-off, system reset, as well as triggering the generation of an alert.

Alert Policies

Alert policies provide a configurable mechanism for configuring and controlling the delivery of an alert to multiple destinations. This enables the creation of 'call down lists' where one alert destination is tried first and then another.

1.6 IPMI Overview

This section presents an overview of IPMI and its main elements and characteristics.

1.6.1 Intelligent Platform Management

The term Intelligent Platform Management refers to autonomous monitoring and recovery features implemented directly in platform management hardware and firmware. The key characteristic of Intelligent Platform Management is that inventory, monitoring, logging, and recovery control functions are available independent of the main processors, BIOS, and operating system. Platform management functions can also be made available when the system is in a powered down state.

Intelligent Platform Management capabilities are a key component in providing enterprise-class management for high-availability systems. Platform status information can be obtained and recovery actions initiated under situations where system management software and normal 'in-band' management mechanisms are unavailable.

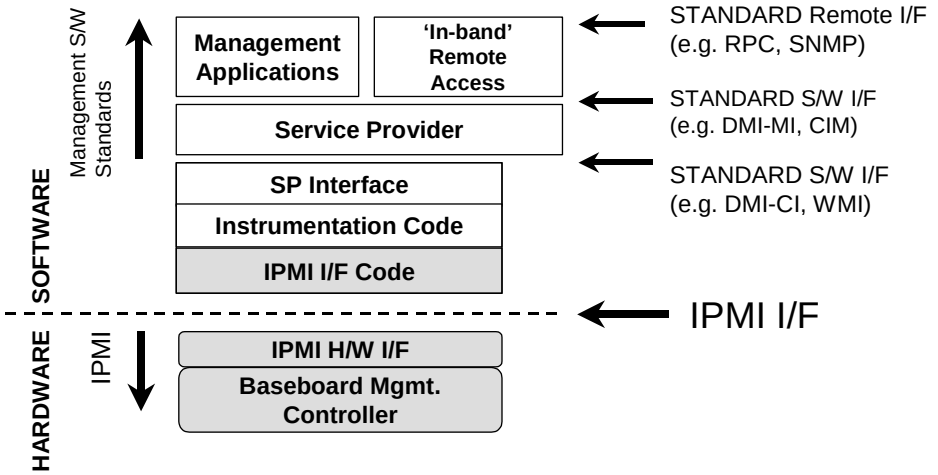
The independent monitoring, logging, and access functions available through IPMI provide a level of manageability built-in to the platform hardware. This can support systems where there is no system management software available for the particular operating system, or the end-user elects not to load or enable the system management software.

1.6.2 IPMI Relationship to other Management Standards

IPMI is best used in conjunction with system management software running under the operating system. This provides an enhanced level of manageability by providing in-band access to the IPMI management information and integrating IPMI with the additional management functions provided by management applications and the OS. System management software and the OS can provide a more sophisticated control, error handling and alerting, than can be directly provided by the platform management subsystem.

IPMI is a hardware level interface specification that is 'management software neutral' providing monitoring and control functions that can be exposed through standard management software interfaces such as DMI, WMI, CIM, SNMP, etc. As a hardware level interface, it sits at the bottom of a typical management software stack, as illustrated in *Figure 1-1*, below.

Figure 1-1, IPMI and the Management Software Stack

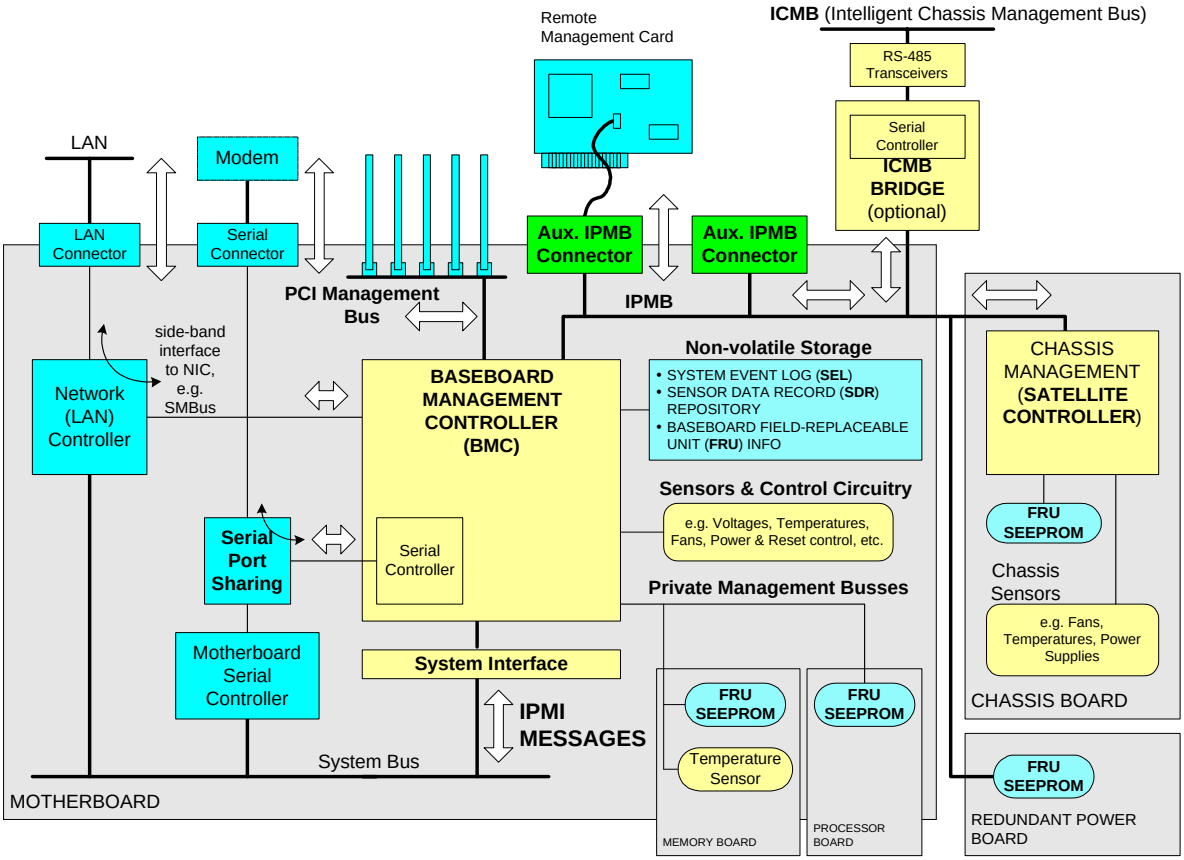


1.6.3 Management Controllers and the IPMB

Figure 1-2, IPMI Block Diagram, shows the main elements of an IPMI implementation. At the heart of the IPMI architecture is a microcontroller called the *Baseboard Management Controller*, or BMC. The BMC provides the intelligence behind Intelligent Platform Management. The BMC manages the interface between system management software and the platform management hardware, provides autonomous monitoring, event logging, and recovery control, and serves as the gateway between system management software and the IPMB and ICMB.

IPMI supports the extension of platform management by connecting additional *management controllers* to the system using the IPMB. The IPMB is an I^2C-based serial bus that is routed between major system modules. It is used for communication to and between management controllers. This provides a standardized way of integrating chassis features with the baseboard. Because the additional management controllers are typically distributed on other boards within the system, away from the 'central' BMC, they are sometimes referred to as *satellite controllers.*

Figure 1-2, IPMI Block Diagram



By standardizing the interconnect, a baseboard can be readily integrated into a variety of chassis that have different management features. IPMI's support for multiple management controllers also means that the architecture is scalable. A complex, multi-board set in an enterprise-class server can use multiple management controllers for monitoring the different subsystems such as redundant power supplies, hot-swap RAID drive slots, expansion I/O, etc., while an entry-level system can have all management functions integrated into the BMC.

IPMI also includes 'low-level' I^2C access commands that can be used to access 'non-intelligent' I^2C devices (devices that don't handle IPMI commands) on the IPMB or *Private Busses* accessed via a management controller. The IPMB can also support SMBus slave devices, with the restriction that the SMB Alert signal is not supported on IPMB, and a controller that implements the IPMB protocol cannot serve as the target for an SMBus *Modified Write Word protocol* transfer from an SMBus slave. Refer to the IPMB and ICMB Specifications (see Reference Documents) for additional information on the IPMB and ICMB.

1.6.4 IPMI Messaging

IPMI uses message-based interfaces for the different interfaces to the platform management subsystem such as IPMB, serial/modem, LAN, ICMB, PCI Management Bus, and the system software-side "System Interface" to the BMC.

All IPMI messages share the same fields in the message 'payload' - regardless of the interface (transport) that they're transferred over. This is represented with the double-ended arrows in *Figure 1-2, IPMI Block Diagram*, The same core of IPMI messages is available over every IPMI-specified interface, they're just 'wrapped' differently according to the needs of the particular transport. This enables a piece of management software that works on one interface to be converted to use a different interface mainly by changing the underlying 'driver' for the particular transport. This also enables knowledge re-use: A developer that understands the operation of IPMI commands over one interface can readily apply that knowledge to a different IPMI interface.

IPMI messaging uses a request/response protocol. IPMI request messages are commonly referred to as *commands.* The use of a request/response protocol facilitates the transfer of IPMI messages over different transports. It also facilitates multi-master operation on busses like the IPMB and ICMB, allowing messages to be interleaved and multiple management controllers to directly intercommunicate on the bus.

IPMI commands are grouped into functional command sets, using a field called the *Network Function Code*. There are command sets for sensor and event related commands, chassis commands, etc. This functional grouping makes it easier to organize and manage the assignment and allocation of command values.

All IPMI request messages have a *Network Function, command,* and optional *data* fields. All IPMI response messages carry *Network Function, command,* optional *data,* and a *completion code* field. As inferred earlier, the differences between the different interfaces has to do with the framing and protocols used to transfer this payload. For example, the IPMB protocol adds fields for I^2C and controller addressing, and data integrity checking and handling, whereas the LAN interface adds formatting for sending IPMI messages as LAN packets.

1.6.5 Sensor Model

Access to monitored information, such as temperatures and voltages, fan status, etc., is provided via the IPMI *Sensor Model*. Instead of providing direct access to the monitoring hardware IPMI provides access by abstracted sensor commands, such as the *Get Sensor Reading* command, implemented via a management controller. This approach isolates software from changes in the platform management hardware implementation.

Sensors are classified according to the type of readings they provide and/or the type of events they generate. A sensor can return either an analog or discrete reading. Sensor events can be discrete or threshold-based.

The different event types, sensor types, and monitored entities are represented using numeric codes defined in the IPMI specification. IPMI avoids reliance on strings for management information. Using numeric codes facilitates internationalization, automated handling by higher level software, and reduces management controller code and data space requirements.

1.6.6 System Event Log and Event Messages

The same approach is applied to the generation and control of platform events. The BMC provides a centralized, non-volatile *System Event Log*, or SEL. Having the SEL and logging functions managed by the BMC helps ensure that 'post-mortem' logging information is available should a failure occur that disables the systems processor(s).

A set of IPMI commands allows the SEL to be read and cleared, and for events to be added to the SEL. The common request message (command) used for adding events to the SEL is referred to as an *Event Message*. Event Messages can be sent to the BMC via the IPMB. This provides the mechanism for satellite controllers to detect events and get them logged into the SEL. The controller that generates an event message to another controller via IPMB is referred to as an *IPMB Event Generator*. The controller that receives event messages is called the *IPMB Event Receiver*.

A generic *Event Receiver* is a controller that accepts a *Platform Event Message* command over whatever media is connected to it, plus internally generated *Event Messages*. The BMC is typically the only generic *Event Receiver* in the system.

Management Controllers that generate Event Messages must know the sensor and event type so it can place that information in the Event Message. This ensures that Event Messages carry important information that can be interpreted without requiring a-priori knowledge of the sensor, or access to the Sensor Data Record for the sensor.

This is often done by 'hardcoding' this relationship into the controller's firmware. However, this approach binds the Sensor Type and Event Type assignment to the generation of event messages. IPMI also includes commands that allow the sensor and event type information to be read from the Sensor Data Record and written into the controller during initialization. This makes it possible to create generic management controllers that do not have to have hard-coded sensor types. For example, a vendor could create a device that provides a number of analog, threshold-based sensors that get assigned as voltage, temperature, or other sensor types according to the type information the system integrator placed in the SDRs for the sensors. An analog input could be assigned as a "+5V" sensor on one system, and a "-12V" sensor on another just by changing the SDRs.

1.6.7 Sensor Data Records & Capabilities Commands

IPMI's extensibility and scalability mean that each platform implementation can have a different population of management controllers and sensors, and different event generation capabilities. The design of IPMI allows system management software to retrieve information from the platform and automatically configure itself to the platform's capabilities. This greatly reduces or eliminates the need for platform-specific configuration of the platform management instrumentation software - enabling the possibility of "Plug and Play" platform-independent instrumentation software.

Information that describes the platform management capabilities is provided via two mechanisms: *capabilities commands* and *Sensor Data Records* (SDRs). Capabilities commands are commands within the IPMI command sets that return fields that provide information on other commands and functions the controller can handle.

Sensor Data Records are data records that contain information about the type and number of sensors in the platform, sensor threshold support, event generation capabilities, and information on what types of readings the sensor provides.

The primary purpose of Sensor Data Records is to describe the sensor configuration of the platform management subsystem to system software. Sensor Data Records *describe* sensors; they do not *instantiate* sensors. For example, adding a new Sensor Data Record does not cause management controller firmware to automatically 'grow' or instantiate a new sensor. But they are used to describe sensors that already exist, and can also be used to tell software to only pay attention to a subset of the available sensors.

Sensor data records have a limited capability to configure *pre-existing* sensors. There is information that an *Initialization Agent* in the BMC to enable or disable sensors and initialize thresholds. This is described more in the following section.

Sensor Data Records also include records describing the number and type of devices that are connected to the system's IPMB, records that describe the location and type of *FRU Devices* (devices that contain field replaceable unit information).

1.6.8 Initialization Agent

SDRs can also hold default threshold values and event generation settings for sensors and management controllers. During system resets, the BMC performs an *initialization agent* function and writes these settings to those sensors that have 'initialization required' field set in their SDR. This eliminates the need for satellite controllers to retain their own non-volatile storage and command interfaces for default settings, and also provides a mechanism to retrigger any events that may have been transmitted before the BMC was ready to accept them. The initialization agent can also be used to assign the Sensor Type to a generic sensor. See *Section 27.6, Sensor Initialization Agent*, for details on the initialization agent process.

1.6.9 Sensor Data Record Repository

Sensor Data Records are kept in a single, centralized non-volatile storage area that is managed by the BMC. This storage is called the *Sensor Data Record Repository* (SDR Repository). Implementing the SDR Repository via the BMC provides a mechanism that allows SDRs to be retrieved via 'out-of-band' interfaces, such as the ICMB, a Remote Management Card, or other device connected to the IPMB. Like most Intelligent Platform Management features, this allows SDR information to be obtained independent of the main processors, BIOS, system management software, and the OS.

1.6.10 Private Management Busses

A Private Management Bus (also referred to as Private Bus) is an I^2C bus that is accessed via a management controller by using IPMI commands for low-level I^2C access. Multiple private busses can be implemented behind a single management controller. IPMI supports using private busses as a mechanism for accessing 24C02-compatible SEEPROMs (Serial Electrically Erasable Programmable ROMs) that hold FRU information. Private busses may also be used to provide low-level access interface for other I^2C or SMBus devices, though the IPMI specification does not cover the way such devices would be used. Each management controller can provide up to eight private busses.

1.6.11 FRU Information

The IPMI specifications include support for storing and accessing multiple sets of non-volatile Field Replaceable Unit (FRU) information for different modules in the system. An enterprise-class system will typically have FRU information for each major system board (e.g. processor board, memory board, I/O board, etc.). The FRU data includes information such as serial number, part number, model, and asset tag.

IPMI FRU information can be made accessible via the IPMB and management controllers. The information can be retrieved at any time, independent of the main processor, BIOS, system software, or OS. This allows FRU information to be retrieved via 'out-of-band' interfaces, such as the ICMB, a Remote Management Card, or other device connected to the IPMB. FRU information can even be available when the system is powered down.

These capabilities allow FRU information to be obtained under failure conditions when FRU access mechanisms that rely on the main processor become unavailable. This facilitates the creation of automated remote inventory and service applications.

IPMI does not seek to replace other FRU or inventory data mechanisms, such as those provided by SM BIOS, and PCI Vital Product Data. Rather, IPMI FRU information is typically used to complement that information or to provide information access out-of-band or under 'system down' conditions.

1.6.12 FRU Devices

IPMI provides FRU information in two ways: via a management controller, or via *FRU SEEPROMs*. FRU information that is managed by a management controller is accessed using IPMI commands. This isolates software from direct access to the non-volatile storage device, allowing the hardware implementer to utilize whatever type of non-volatile storage they want.

In order to more economically support providing FRU information on multiple platform modules, IPMI also allows simple 24C02-compatible SEEPROM (Serial Electrically Erasable Programmable ROM) chips to be used for storing FRU information. ('24C02'-type devices are non-volatile storage devices that have a built-in I^2C-compatible interface).

FRU SEEPROMs provide a mechanism for implementing FRU information without requiring a management controller on the field replaceable unit. FRU SEEPROMs can be accessed via a Private Management Bus connected to a management controller, or if necessary, can be placed directly on the IPMB or PCI Management Bus. While supported, it is generally recommended that devices with I^2C/SMBus interfaces that lack data integrity checks (e.g. checksums), such 24C02-type SEEPROMs, are not placed on 'public' busses such as IPMB and PCI-SMBus. This is because without data integrity checks it is possible that a misbehaved third-party add-in device could cause a bus 'glitch' that would result in an undetected error when reading or writing the SEEPROM. (Note: depending on the type of device, I^2C addressing places a limit on the number of devices that can be placed directly on the IPMB. Refer to the *IPMB I^2C Address Allocation* specification for more information.)

1.6.13 Entity Association Records

Entity Association Records are a special type of SDR that provides a definition of the relationships among platform entities. For example, an Entity Association can be set up that groups a set of individual power supplies into a redundant power unit. A 'redundancy lost' event on the power unit can then be correlated with the individual power supply failure. Without the Entity Association information, the 'redundancy lost' and 'power supply failed' events would be disjoint events that could only be correlated based on a-priori knowledge of the system.

1.6.14 Linkage between Events and FRU Information

Included in the SDRs is information that indicates which system entity a sensor is monitoring (e.g. a memory board) and also provide a link to the FRU information for the entity. SDRs use a set of codes that specify which controller holds the sensor, the sensor type (e.g. temperature), the particular instance of the sensor (e.g. sensor #2), the sensor's event and reading type (e.g. discrete or threshold-based), the set of events it can generate, and associated bit fields that indicate which specific events a sensor can produce.

The same codes and bit fields directly map to the information that is passed in event messages and logged in the SEL. Thus, a SEL entry can indicate the controller, sensor, sensor type, and event type associated with the event. This information provides a useful level of information by itself - but when combined with SDR information, the event can be correlated to the entity and FRU associated with the event. Correlating an event to the FRU can help guide a service person to the problem area, or even be used to identify the replacement parts they should bring to a site.

1.6.15 Differentiation and Feature Extensibility

Platform management features continue to evolve. While IPMI seeks to provide a standardized interface to cover the majority of platform management needs, explicit provisions have been made throughout IPMI to support OEM differentiation and new features. Special ranges of code values and commands have been reserved to allow OEM sensors, events, and value-added functions to be implemented within the IPMI framework.

1.6.16 System Interfaces

IPMI defines three standardized system interfaces that system software uses for transferring IPMI messages to the BMC. In order to support a variety of microcontrollers, IPMI offers a choice of system interfaces. Using these interfaces is key to enabling cross-platform software. The system interfaces are similar enough so that a single driver can be created that supports all IPMI system interfaces.

The system interface connects to a system bus that can be driven by the main processor(s). The present IPMI system interfaces are I/O mapped. Any system bus that allows the main processor(s) to access the specified I/O locations, and meet the timing specifications, can be used. Thus, an IPMI system interface could be hooked to the ISA-bus, X-bus, or a proprietary bus off the baseboard chip set.

The three IPMI system interfaces are:

Keyboard Controller Style (KCS) — The bit definitions, and operation of the registers follows that used in the Intel 8742 Universal Peripheral Interface microcontroller. The term 'Keyboard Controller Style' reflects the fact that the 8742 interface was used as the legacy keyboard controller interface in PC architecture computer systems. This interface is available built-in to several commercially available microcontrollers. Data is transferred across the KCS interface using a per-byte handshake.

System Management Interface Chip (SMIC) — The SMIC interface provides an alternative when the implementer wishes to use a microcontroller for the BMC that does not have the built-in hardware for a KCS interface. This interface is a three I/O port interface that can be implemented using a simple ASIC, FPGA, or discrete logic devices. It may also be built-in to a custom-designed management controller. Like the KCS interface, a per-byte handshake is also used for transferring data across the SMIC interface.

Block Transfer (BT) — This interface provides a higher performance system interface option. Unlike the KCS and SMIC interfaces, a per-block handshake is used for transferring data across the interface. The BT interface also provides an alternative to using a controller with a built-in KCS interface. The BT interface has three I/O-mapped ports. A typical implementation includes hardware buffers for holding upstream and downstream message blocks. The BT interface can be implemented using an ASIC or FPGA, or may be built-in to a custom-designed management controller.

1.6.17 Other Messaging Interfaces

In addition to the System Interface and IPMB, IPMI messaging can be carried over other interfaces, such as LAN, serial/modem, ICMB, and PCI management bus. IPMI includes a communication infrastructure that supports transferring messages between these interfaces as well as to the BMC.

1.6.18 LAN Interface

The LAN interface specifications define how IPMI messages can be sent to and from the BMC encapsulated in RMCP (Remote Management Control Protocol) packets datagrams. This capability is also referred to as “IPMI over LAN”. IPMI also defines the associated LAN-specific configuration interfaces for setting things such as IP addresses other options, as well as commands for discovering IPMI-based systems.

The Distributed Management Task Force (DMTF) specifies the RMCP format. This same packet format is used for non-IPMI messaging via the DMTF’s Alert Standard Forum “ASF” specification. Using the RMCP packet format enables more commonality between management applications that operate in an environment that includes both IPMI-based and ASF-based systems. More information on IPMI and ASF is provided below.

1.6.19 Serial/Modem Interface

The Serial/Modem Interface specifications define how IPMI messages can be sent to and form the BMC via a direct serial or external modem connection. The specification supports three *connection modes* that define the protocol for delivering IPMI messages via serial/modem:

- **Basic Mode:** The IPMI messages are encapsulated with minimal additional framing and escaping for transport over a serial/modem connection. Basic Mode provides the highest performance but requires an ‘IPMI-aware’ serial application.
- **PPP Mode:** The IPMI messages are encapsulated in the same RMCP format as used for LAN messages, but are delivered via a PPP connection. PPP mode allows remote applications to take advantage of built-in PPP support in the OS for things such as dialing and authentication, and provides the highest commonality with LAN-based software, but at the cost of lower throughput.
- **Terminal Mode:** Terminal Mode defines how IPMI messages can be transferred using printable characters. It also includes a limited number of English ASCII text commands for doing such things as getting a high level system status and causing a system reset or power state change. Terminal mode is lower performance than Basic Mode and more limited in capabilities than both Basic Mode and PPP Mode, but offers a mechanism for those who are transitioning to IPMI and more sophisticated interfaces from a legacy, character-based environment

1.6.20 IPMI and ASF

IPMI and ASF are complementary specifications that can provide platform management in a ‘pre-boot’ or ‘OS absent’ environment. IPMI uses a management microcontroller as the main element of the management system, whereas ASF presently focuses on having an ‘alert sending device’ - typically the network controller - polling devices on the motherboard and autonomously generating alerts. As of this writing, ASF’s scope primarily covers the way an alert sending device polls sensor devices and sends alerts, and the specification of ‘LAN’ commands for discovering RMCP-based systems and performing emergency reset and power off actions.

This includes the supporting specification of SMBus interfaces to ‘ASF Sensor Devices’ that can be polled by the alert sending device, the specification of the RMCP packet format, and the specification of SMBus-based commands that can be used to send a ‘push’ alert via an alert sending device.

While somewhat of an oversimplification, ASF may be considered to be scoped for ‘desktop/mobile’ class systems, and IPMI for ‘servers’ where the additional IPMI capabilities such as event logging, multiple users, remote authentication, multiple transports, management extension busses, sensor access, etc., are valued. However there are no restrictions in either specification as to the class of system that the specification can be used. I.e. you can use IPMI for desktop and mobile systems and ASF for servers if the level of manageability fits your requirements.

IPMI and ASF share a number of formats, data structures, and enumerations. It is expected that this will continue to grow.

- Shared management packet format: IPMI uses ASF 'RMCP' packet format for delivering IPMI messages over LAN and PPP and ASF messages for LAN discovery. The RMCP format includes a message class explicitly for IPMI use.
- Common LAN Alert Format: Both generate LAN Alerts using the IPMI PET (Platform Event Trap) Specification for SNMP Traps
- Common Flags for boot control: IPMI uses a superset of the boot flags defined in ASF.
- Common enumerations for sensor types and event types: ASF uses the IPMI enumerations for sensor and event types. These values are used in Alerts and ASF Sensor Device Status.
- Common BIOS progress codes: IPMI uses ASF BIOS Error and Progress codes.
- Hardware: IPMI management controllers and ASF alert sending devices can both use ASF Sensor Devices. In an IPMI application these can be place on private management busses and polled by the BMC, they can also be used on the PCI management bus. In an ASF application, the devices would typically always be on the PCI management bus or main SMBus and polled by the Network Controller(s).

1.6.21 LAN Alerting

IPMI supports LAN Alerting in the form of SNMP Traps that follow the Platform Event Trap (PET) format. (Refer to [PET] for more information.) SNMP Traps are typically sent as unreliable datagrams. However, IPMI includes a PET Acknowledge and retry options that allows an IPMI-aware remote application to provide a positive acknowledge that the trap was received.

1.6.22 Serial/Modem Alerting and Paging

The IPMI specification supports several options for alerting over a serial/modem connection:

- Dial Page: Sending a numeric page by using an external modem to generate 'touch tones'.
- TAP Page: The BMC connects to a TAP 1.8 paging service and delivers an alphanumeric page.
- PPP Alert: The BMC connects to a remote LAN via PPP and delivers a PET trap to a specified IP address.

1.6.23 Platform Event Filtering (PEF)

Platform Event Filtering (PEF) provides a mechanism for configuring the BMC to take selected actions on event messages that it receives or has internally generated. These actions include operations such as system power-off, system reset, as well as triggering the generation of an alert.

The BMC maintains an *event filter* table that is used to select which events trigger an action and which actions to perform. Each time the BMC receives an event message (either externally or internally generated) it compares the event data against the entries in the event filter table. The BMC scans all entries in the table and collects a set of actions to be performed as determined by the entries that were matched.

1.6.24 Call Down Lists and Alert Policies

The IPMI specification allows an implementation to support configurable alert policies that determine how an alert will be processed. These can be used to create a 'call down list' of different destinations that an alert gets sent to. Alert policies can have destinations of different types and on different channels. For example, a policy could be defined to first try to send an alert to LAN address 'A', and if that fails send it to LAN address 'B', and then send a Dial Page via the modem, and if that fails, a TAP page.

IPMI allows the alert destinations to be configured in any order. I.e. you can pick whether an alert goes out via serial/modem first, or via LAN first. The main limitation comes from the number of policy entries that a given implementation supports.

1.6.25 Channel Model, Authentication, Sessions, and Users

IPMI v1.5 incorporates a common communication infrastructure referred to as the 'Channel Model'. This is an extension of the channels that were used as part of messaging in IPMI v1.0.

Channels provide the mechanism for directing the routing of IPMI messages between different media connections to the BMC. A *channel number* identifies a particular connection. For example, 0 is the channel number for the primary IPMB. Up to nine total channels can be supported (the System Interface and primary IPMB, plus seven additional channels with a media type assigned by the implementer.) Channels can thus be used to support multiple IPMB, LAN, Serial, etc., connections to BMC.

Channels can be *session-based* or *session-less.* A *session* is used for two purposes: As a framework for user authentication, and to support multiple IPMI Messaging streams on a single channel. Session-based channels thus have at least one user 'login' and support user and message authentication. Session-less channels do not have users or authentication. LAN and serial/modem channels are examples of session-based, while the System Interface and IPMB are examples of session-less channels.

In order to do IPMI messaging via a session, a session must first be *activated.* The act of activating a session is one of authenticating a particular user. This is accomplished using a 'challenge/response' mechanism, where a challenge is requested using a *Get Session Challenge* command, and the signed challenge returned in an *Activate Session* command.

The specification supports different algorithms for the signature - these are referred to as Authentication Types. Authentication Types include 'none', 'straight password', the MD2 and MD5 message-digest algorithms, etc. For consistency, session-based channels always use the *Get Session Challenge* and *Activate Session* commands even if Authentication Type is 'none'. (In this case, dummy values are used for the signatures.)

A session has a *Session ID* that is used for tracking the state of a session. The Session ID mechanism allows multiple sessions to be able to be simultaneously supported on a channel.

The message signature, Session ID, and other session related information is separate from the actual IPMI message content. Thus, a packet carrying an authenticated IPMI message can be thought of as being comprised of a 'Session Packet' that includes the session-specific fields and carries an IPMI message as its payload.

The concept of *user* is essentially a way to identify a collection of privilege and authentication information. User configuration is done on a *per channel* basis. This means that a given user could have a different password and set of privileges for accessing the BMC via a LAN channel than via a serial channel.

Privilege Levels determine which IPMI commands a given user can execute over a given channel.

Privilege Limits set the maximum privilege level that a user can operate at. A user is configured with a given maximum privilege limit for each channel. In addition there is a *Channel Privilege Limit* that sets the maximum limit for all users on a given channel. The *Channel Privilege Limit* takes precedence over the privilege configured for the user. Thus, a user can operate at a privilege level that is no higher than the lower of the User Privilege Limit and the Channel Privilege Limit.

1.6.26 Standardized Watchdog Timer

Watchdog Timer capabilities have been commonly deployed in Enterprise-class servers. IPMI provides a standardized interface for a system Watchdog Timer. This timer can be used for BIOS, OS, and OEM applications. The timer can be configured to automatically generate selected actions when it expires, including power off, power cycle, reset, and interrupt. The timer function can also automatically log the expiration event. Setting '0' for the timeout interval allows the timeout action to be initiated immediately. This provides a means

for devices on the IPMB, such as Remote Management Cards, to use the Watchdog Timer to initiate emergency reset and other recovery actions dependent on the capability of the timer.

1.6.27 Standardized POH Counter

This is an optional counter to return a counter value proportional to the system operating (S0) power-on hours.

1.6.28 IPMI Hardware Components

IPMI provides very few restrictions on the actual hardware components used to implement the platform management hardware. IPMI seeks to 'standardize the interface, not the implementation'. IPMI was designed so that it can be implemented with 'off-the-shelf' components. Thus, IPMI does not require specific microcontrollers to be used for management controllers, nor special ASICs or proprietary logic devices. As long as the interface, timing and (in the case of IPMB and ICMB) electrical specifications are met, the choice of components is up to the implementer. It is mandatory to implement a system interface that is compatible with one of the three specified system interfaces.

1.7 IPMI and BIOS

The level of interaction between BIOS and IPMI is greatly dependent on the implementation and number of optional capabilities that are to be supported. It is possible to have an IPMI implementation that does not require any BIOS support, other than that required to meet any applicable ACPI or Plug 'N Play requirements for reporting the I/O and/or interrupt resources used by the IPMI system interface.

In some implementations, BIOS may be responsible for the initialization or startup of certain functions in the management controllers, such as setting the initial timestamp time in the SEL and/or SDR devices. BIOS may also perform tests of the platform management hardware and management controllers during POST.

It is recommended that BIOS include provisions for checking and reporting on the basic health of BMC by executing the *Get Self Test Results* command and checking the result.

It's expected that most implementations will provide BIOS features that take advantage of IPMI. For example, it is expected that many implementations will use IPMI to log POST errors, or to log 'system boot' events so that events can be tracked relative to the last boot time. Another expectation is that many systems utilize the IPMI Watchdog Timer function with BIOS.

With IPMI v1.5, the BIOS can share in additional capabilities. For example, IPMI v1.5 defines a new LAN-based interface. The BIOS can help keep the BMC updated with the LAN IP Address assignment. IPMI v1.5 also includes a serial/modem interface with support for a capability called 'serial port sharing' in which the serial controller can be shared between the BMC and BIOS-based serial console redirection. There is also a set of 'boot flags' that BIOS can read to direct its operation following a system management initiated reset, power cycle, or power up.

1.8 System Management Software (SMS)

The Management Controllers, Sensors, SEL information, SDR information, etc., are of limited value without System Management Software to interpret, handle, and present the information. Platform management is only a subset of systems management. System Management Software takes platform management information and links it into other aspects of systems management, such as software management and distribution, alerting, remote console access, etc.

With respect to the platform management architecture and this specification, System Management Software:

- Polls the System Event Log for new Event information, acting on it as appropriate. This may include taking actions such as sending alerts on the network, presenting a local ‘pop-up’ message, shutting down an application, consolidating the information with other ‘system log’ data, etc. System Critical Events are primarily communicated to system management software using the System Event Log as a 'mailbox' between the originator of the Event Message and the system management software.

- Manages the System Event Log. The SEL may contain ‘critical’ event information that should not be lost. Therefore, the SEL device will not automatically clear the SEL if it gets full. This operation is based on the assumption that it is the first events that are most indicative of the root cause of a problem, and that later events may be ‘side-effects’ which, if the SEL were implemented as a ‘FIFO’ could cause the ‘root cause’ events to get lost. Instead, System Management Software has the responsibility for determining when SEL entries should be cleared. System Management Software can migrate the SEL contents to disk, to the system’s event log, or even to remote storage as desired.

- Reads and interprets the SDR Repository information. System Management Software uses this information to determine the sensor population and capabilities. The Sensor Data information can also be presented to provide a description of the system’s manageability features.

- ‘Polls’ sensors. System Management Software takes the SDR information and uses it to access the sensors, either in a polled or ‘on demand’ basis, to monitor and present the system’s current health and status. Note that whenever possible, System Management Software should rely on event generation for detecting error conditions, and avoid the overhead associated with polling. ‘Normal’ health status does not generally need to be polled, but would be delivered ‘on demand’.

- Potential Event Message Source. System Management Software can also send Event Messages to get events added to the System Event Log. This allows SMS to record information that may be required for ‘post-mortem analysis’ should it become necessary for System Management Software to shut-down, power-cycle, reset, or otherwise ‘off line’ the system as a response to a system event. The SEL should be reserved for ‘critical’ hardware-related errors. The majority OS and software errors should not be written to the SEL. Candidate errors for the SEL are errors that block normal ‘in-band’ management mechanisms.

1.9 SMI Handler

Not all platform management events come through management controllers or from system software. Some events come from baseboard interrupts. This may include platform events such as correctable and uncorrectable ECC errors, critical NMIs (Non-maskable Interrupts) such as PCI PERR (parity error), PCI SERR (system error), bus timeout interrupts, etc. In some implementations, the platform management hardware maps these ‘critical interrupts’ to the system SMI (System Management Interrupt) signal. The SMI Handler runs, and, as part of handling these critical interrupts, generates an Event Message to cause the event to get logged in the SEL. The SMI Handler can also take autonomous, ‘emergency’ action, such as powering off or resetting the system, or propagating an NMI to the operating system.

The SMI Handler is typically a routine that is loaded and initialized into a protected area of memory by the BIOS. SMI is the highest priority non-maskable interrupt in the system. When asserted, it switches the processors into ‘System Management Mode’ (SMM). Upon entry into SMM, the processor state is saved and a memory configuration is entered where the SMI Handler has full access to system memory and I/O space. This allows the SMI Handler to implement its management functions in an OS-independent manner. The key aspect to this being that the SMI Handler code will run even if the OS is ‘hung’. This makes it ideal for implementing certain critical and emergency management functions.

The explicit interface and functionality between an SMI Handler and the BMC is implementation dependent and is not covered by this specification. The implementation of system-specific communication interfaces can be aided using the OEM bits and flags in the BMC-system interface commands.

1.10 Overview of Changes from IPMI v1.0

This section assumes familiarity with the IPMI v1.0 specification. If you're new to IPMI, you can skip ahead to the next section. This is not intended to be a complete "to the bit" list of all the changes, but is provided as a guide for understanding what's involved in moving to support IPMI v1.5.

- Most commands that have version numbers had their version numbers rev'd to 51h for IPMI 1.5
- The *Get Device ID* command was extended a couple of optional firmware revision bytes, per NEC request. Just display them as hex-ASCII.
- The IPMI sensor commands are the same as in v1.0, though a small amount of typo corrections and additional clarifications have been made in those sections.
- The IPMI watchdog commands are backward compatible with IPMI v1.0. A previously reserved bit has been defined as a new 'don't stop' bit that allows the watchdog timer to be reconfigured without stopping it.
- The IPMI v1.0 event commands are the same. A couple of new event commands, *Set/Get Last Processed Event* have been added to allow someone using the new IPMI v1.5 PEF capability to set or determine whether or not PEF has pending events to process.
- Sensor Event/Reading Type codes - The POST Error sensor is now called "System Firmware Progress" and includes new offsets for POST errors and progress the follow the DMTF ASF specification. There's also a new 'Management Subsystem Health' sensor type, 28h. Please check the Sensor Event/Reading Type code table for any other changes.
- The SEL Event Record format is the same except that four previously reserved bits now hold a channel number, and the SEL Record version "EvMRev" field goes from 3h to 4h. It's possible that two events would be identical except for the channel number field. Software that handles or displays events should interpret the channel number field in order to differentiate between events coming from different channels.
- The SDRs are the same with the exception of the version number and a field changes to accomodate a necessary fourth bit for the channel number. This change affected SDRs 01h, 02h, 10h, 11h and 12h. The addition of a channel number in the Type 12h SDR caused the two bytes following to get pushed down.
- The Entity Instance value in the Entity Association Record has been split into two ranges: one for 'system relative' IDs and another for 'device relative' IDs. Most implementations will be able to use their existing Entity Instance assignments since the lower range of values are for the 'system relative' Entity Instance values, which map to the IPMI v1.0 definition of the Entity Instance value.
- SDR Type 14h is being deprecated. IPMI 1.5 systems and software should not use SDR Type 14h. Software should use the new *Get Channel Info* command instead.
- The SEL Event Record format is the same except that four previously reserved bits now hold a channel number, and the SEL Record version "EvMRev" field changes from 3h to 4h.
- The IPMB message format remains the same.
- The *Send Message* command is backward compatible with v1.0 with respect to using it to access the IPMB. I.e. you don't need to make any changes to access the IPMB.
- The *Master Write/Read* I^2C has had the "I^2C" dropped from the name. It is now the *Master Write/Read* command. This command is backward compatible with IPMI v1.0. Reserved bits 7:4 in byte one have become a Channel ID, but 0h is when accessing the IPMB or private management busses as in IPMI v1.0. A non-zero channel value would only be used for accessing additional IPMBs or a PCI Management Bus.
- The read/write FRU commands are the same.

2. Logical Management Device Types

The Intelligent Platform Management architecture is comprised of a number of 'logical' management devices. These are implemented by and within the 'physical' system elements such as the management controllers, I^2C bus, system ASICs, etc.

Each 'logical device' type represents the definition of a particular set of mandatory and optional commands. For the purposes of this specification, the logical management devices are:

IPM Device — Intelligent Platform Management device. This represents the 'basic' intelligent device that responds to the platform sensor and event interface messages. All Intelligent Platform Management devices on the IPMB are expected to respond to the mandatory 'IPM Device' commands. These are also referred to as the 'global' commands. Management Controllers that communicate via compatible messages to the system are also considered IPM devices.

Sensor Device — The Sensor Device is a device that provides the command interface to one or more sensors. Sensor Devices provide a set of commands for discovering, configuring and accessing sensors.

SDR Repository Device — The SDR Device is the logical management device that provides the interface to the Sensor Data Records (SDR) for the system. The SDR Device provides a set of commands for storing and retrieving Sensor Data Records.

SEL Device — The SEL Device is the logical management device that provides the interface to the System Event Log for the system. The SEL Device provides a set of commands for managing the System Event Log.

FRU Inventory Device — The FRU Inventory Device provides the interface to a particular module's FRU Inventory (serial number, part number, asset tag, etc.) information. There will typically be one set of FRU Inventory information for each major module in the system. There can be just as many FRU Inventory Devices providing access to that information. The *Primary FRU Inventory Device* for a given management controller is defined as the device that contains the information about the FRU that holds the management controller itself.

Event Receiver Device — The Event Receiver Device accepts and acknowledges Event Request Messages. The normal action for the Event Receiver Device is to then pass the Event Message to the SEL Device for logging. An *IPMB Event Receiver* refers to an Event Receiver that accepts Event Messages from the IPMB.

Event Generator Device — The Event Generator Device represents the functionality that is used to deliver Event Messages to the Event Receiver Device. The Event Generator Device includes commands to allow configuration of Event Message delivery. The term *IPMB Event Generator* refers to the capability to generate an Event Message on the IPMB. The BMC is typically an IPMB Event Receiver, but not an IPMB Event Generator.

Application Device — A physical instantiation of an Intelligent Platform Management device will most likely have some 'device specific' functionality that it implements that falls outside the 'standard' sensor and event functions. This functionality is referred to as the devices 'Application' functionality. Commands that address this functionality are viewed as being handled by an 'Application' logical device.

PEF Device This logical device represents the functions associated with comparing an event message against a set of selectable ‘event filters’ and generating a selectable action on a match.

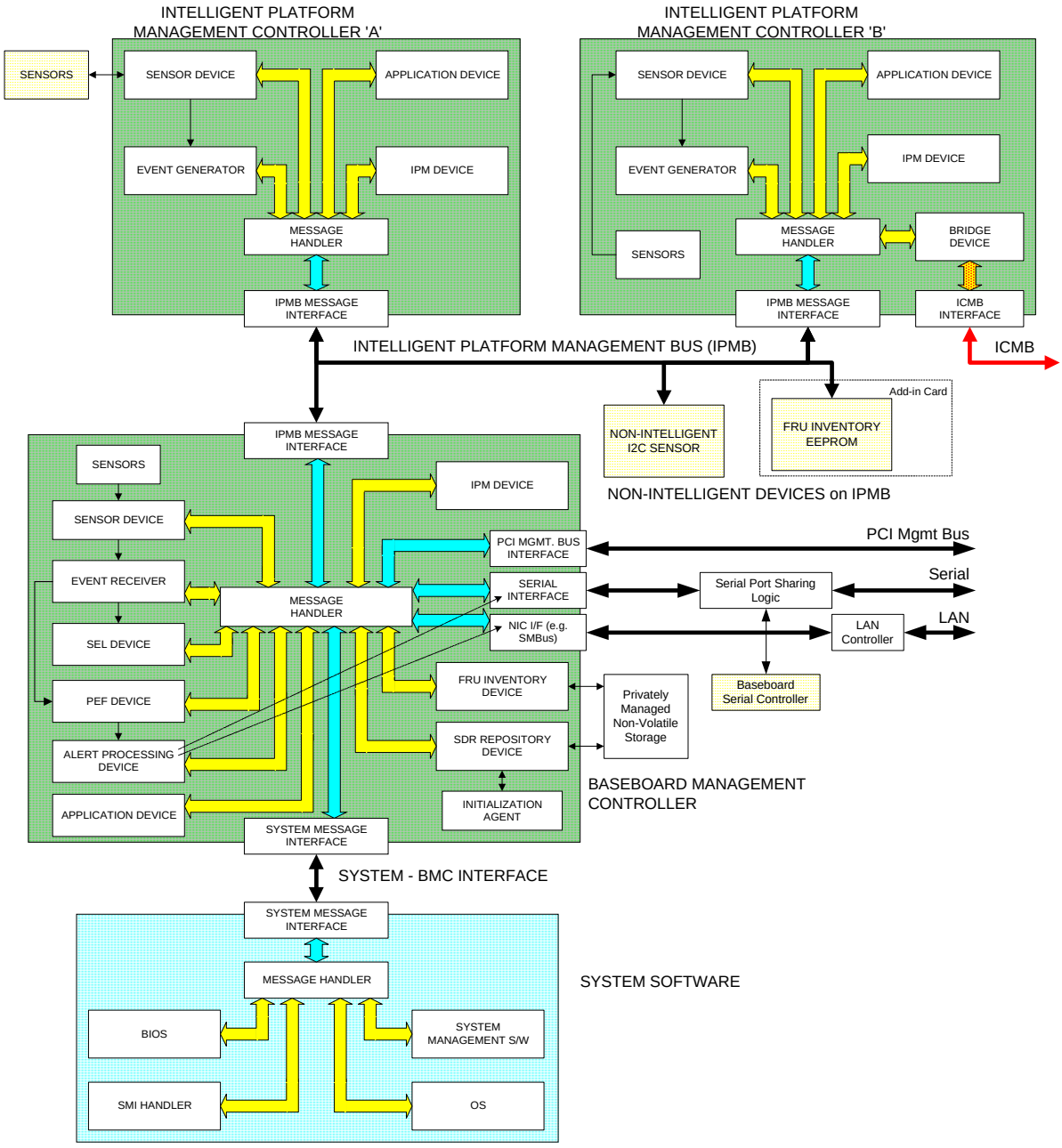
Alert Processing Device This logical devices represents the functions associated with queuing up and processing alerts, and alert policies that determine which destinations an alert will be sent to.

Chassis Device This chassis control device represents functions associated with recovery control actions such as power on/off, power cycle, reset, diagnostic interrupt, chassis identification indicator, and system boot.

Message Handler This logical device represents the functions associated with configuration and operation of message authentication and routing, both internal to the BMC and among the different interfaces to the BMC.

The Intelligent Platform Management Bus can be considered as defining other ‘logical’ devices as well, such as the ‘Bridge Device’ for the Intelligent Chassis Management Bus (ICMB). Refer to the *Intelligent Platform Management Bus Protocol Specification* for more information.

Figure 2-1, Intelligent Platform Management Logical Devices



3. Baseboard Management Controller (BMC)

The management architecture can be implemented by centralizing the most common functions into a 'central' management controller in the system. This controller is often called the *Baseboard Management Controller,* or BMC. In some system implementations, the BMC may be the only management controller. The BMC typically provides the following platform management functions:

System Interface — The BMC provides the System Interface to the IPMI-based platform management subsystem. The System Interface is the interface through which system software sends and receives messages to and from the BMC.

Message Handler — The BMC provides functions for routing messages between the different interfaces, including the System Interface, IPMB, serial/modem, LAN, etc. The Message Handler may also be thought of as where shared messaging functions for configuring channel characteristics and user privileges reside.

SEL Interface — The BMC provides the interface to the System Event Log (SEL). The BMC allows the SEL to be accessed both from the system side, but also from the Intelligent Platform Management Bus and other external interfaces to the BMC.

Event Generator — The BMC itself will typically be responsible for monitoring and managing the system board. For example, monitoring baseboard temperatures and voltages. As such, the BMC will also be an Event Generator *internally*, sending the Event Messages that it generates internally to its Event Receiver functionality. Note the BMC is not typically and *IPMB Event Generator*. That is, it does not typically issue Event Messages onto the IPMB.

SDR Repository Interface — The BMC will also provide the interface to the SDR (Sensor Data Record) Repository. As with the System Event Log, the BMC allows the records in the SDR Repository to be accessed either via the Intelligent Platform Management Bus or via the system interface.

IPMB Interface — A BMC will typically support an IPMB connection. The IPMB enables the BMC to accept IPMI request messages from other management controllers in the system. The IPMB provides a simple integration point for connecting the 'chassis' management features to the baseboard management. The IPMB can also provide a connection that enables add-in cards to get access to the platform management subsystem.

A BMC that includes IPMB Interface support also provides the capability for system software to send and receive messages to and from the IPMB using the BMC as a kind of communication controller.

IPMB Event Receiver — When an IPMB is implemented, the BMC serves as the primary IPMB Event Receiver for the system. Event Messages can be sent to the BMC from the system or from other controllers the IPMB.

Private Bus Controller — FRU SEEPROMs may be provided on Private Management Busses behind the BMC. The BMC can server as a communication controller that provides access to Private Management Busses and provide access to FRU SEEPROMs and other non-intelligent devices via the *Master Write-Read* command.

FRU Information Interface — The BMC provides access to FRU information for the base system board. The FRU information for the board holding the BMC is obtained by sending *FRU Commands* to the BMC's LUN 00b.

OEM Commands — A BMC implementation can include special support for OEM-unique features and

functions. One way of accomplishing this is by implementing OEM commands through the IPMI messaging interfaces.

Watchdog Timer

IPMI defines common command interfaces for configuring and accessing a watchdog timer function in the BMC. This timer can be used as an aid in monitoring the health of BIOS and system software. The watchdog timer can be used by different types of software such as BIOS, pre-boot, OS, and system management software. Once started the timer must be periodically reloaded by software in order to keep it from expiring. If software ceases to run, the timer will expire and generate a timeout action.

The IPMI definition allows different actions to be selected to occur on a watchdog timeout. This includes reset, power off, power cycle, etc. and a 'pre-timeout interrupt' option that, if provided, can be used to generate a system interrupt shortly before the timeout. The definition includes 'timer use' fields that keep track of what type of software (BIOS, OS, System Management Software, etc.) started the timer. The timeout action and 'timer use' information can be automatically logged to the SEL when the timeout occurs. This provides a record of when the timeout occurred, what software was using the timer, and what action was taken.

In addition, a BMC may implement additional functions for messaging and alerting, including:

Serial/Modem Interface The BMC can provide a serial/modem interface that allows it to receive IPMI messages over a serial connection to the BMC.

Serial Port Sharing Serial Port Sharing is a separate capability that works in conjunction with the serial/modem interface. Serial Port Sharing provides a mechanism where the BMC can control logic that allows a single serial connector to be shared between a serial controller on the baseboard and a serial controller for the BMC.

LAN Interface From the IPMI point-of-view, the interface to the network controller is dedicated to the BMC. That is, there are no special commands for coordinating the sharing of the network controller between system software access and BMC access, as there are with Serial Port Sharing. If the network controller is shared between system software and the BMC, this is generally accomplished via special hardware in the network controller that enable BMC traffic and system traffic to be interleaved.

PCI Management Bus Interface The BMC can implement a PCI Management Bus Interface that enables the BMC to accept IPMI request messages from add-in cards that plug into a PCI slot. The PCI management bus and IPMB can serve complementary roles. The IPMB providing a mechanism for integrating management functions between baseboard and chassis board functions, while the PCI Management Bus connection can be used to support add-in cards. This division allows the inter-board management communications to be kept separate from add-in card communications.

Platform Event Filtering (PEF) Platform Event Filtering is an ability for the BMC to perform a configurable action based on an event, by matching the event against a set of ‘event filters’. The actions that a BMC can elect to implement include power off, reset, power cycle, generate diagnostic interrupt, and send an alert.

Alert Processing IPMI v1.5 supports the ability for a BMC to deliver alerts such as SNMP Traps in the Platform Event Trap (PET) format, over media such as LAN and PPP, plus the ability to perform numeric and/or alphanumeric paging via a serial/modem connection. Alert processing includes the ability to support sending alerts to multiple destinations, and to cluster destinations into sets called ‘Alert Policies’. Enabling alert policies with PEF makes it possible to configure the system so critical events are delivered to destinations in a ‘high priority’ alert policy, while non-critical events would go to destinations in a ‘low priority’ alert policy.

3.1 Required BMC Functions

The following table summarizes the major required and optional functions for an IPMI-conformant BMC.

Table 3-1, Required BMC Functions

Function	M/O	Description
IPM Device	M	The BMC must implement the mandatory IPM Device commands. If an IPMB is provided, the mandatory commands must be accessible from the IPMB unless otherwise noted.
System Interface	M	The implementation must provide BMC access via one of the specified IPMI system interfaces.
SDR Repository	M	The BMC must provide a SDR Repository to hold Sensor, Device Locator, and Entity Association records for all sensors in the platform management subsystem. This does not need to include SDRs for sensors that only generate events. It is recommended that at least 20% additional space is provided for platform management extensions. The SDR Repository must be accessible via the system interface. If an IPMB is provided, the SDR Repository must be readable via that interface as well. SDR update via the IPMB interface is optional. SDR Repository access when the system is powered up or in ACPI 'S1' sleep is mandatory, but access when the system is powered-down or in a >S1 sleep state is optional.
IPMB Interface	O	The IPMB is highly recommended, but optional. The BMC must provide the system interface to the IPMB. If an IPMB is implemented, at least one of the specified IPMB connectors must be provided. Refer to the IPMB Protocol specification for connector definition. In addition the BMC must implement a message channel that allows messages to be sent from the IPMB to the system interface, and vice-versa, and any other mandatory IPMB support functions and commands.
Watchdog Timer	M	The BMC must provide the standardized Watchdog Timer interface, with support for system reset action. Certain functions within the Watchdog Timer are optional. Refer to the sections on the Watchdog Timer for information.
Event Receiver	M	The BMC must implement an Event Receiver function and accept Event Messages via the system interface. If an IPMB is provided, the Event Receiver function must also accept Event Messages from the IPMB. Event Receiver operation while the system is powered up or in ACPI 'S1' sleep is mandatory, but operation when the system is powered down or in a >S1 sleep state is optional.
SEL Interface	M	The BMC must provide a System Event Log interface. The event log must hold at least 16 entries. SEL access must be provided via the system interface. If an IPMB is provided, the SDR Repository must be accessible via that interface as well. SDR Repository access when the system is powered up or in ACPI 'S1' sleep is mandatory, but access when the system is powered-down or in a >S1 sleep state is optional.
FRU Inventory	M (v1.5)	The BMC must provide a logical Primary FRU inventory device, accessible via the *Write-* and *Read FRU Data* commands. The *FRU Inventory Device Info* command must also be supported. It is highly recommended that all other management controllers also provide a Primary FRU inventory device. (This was optional in IPMI v1.0.)
Initialization Agent	M	The initialization agent function is one where the BMC initializes event generation and sensors both internally and on other management controllers according to initialization settings stored in the SDR for the sensor.
Sensors	O	The BMC can provide sensors. A typical server BMC would provide sensors for baseboard temperature, voltage, and chassis intrusion monitoring.
Internal Event	M	The BMC must generate internal events for the Watchdog Timer. It is highly

Function	M/O	Description
Generation		recommended that sensors generate events to eliminate the need for system management software to poll sensors, and to provide “post-mortem” failure information in the SEL. Internal event generation for sensors is optional, but highly recommended - particularly for ‘environmental’ (e.g. temperature and voltage) sensors.
External Event Generation	O	The BMC could be designed to accept the *Set Event Receiver* command to allow it to be set as an IPMB Event Generator and send its event messages to another management controller. This would primarily be used for development and test purposes.
PCI Management Bus Interface	O	The BMC supports a connection to a PCI Management bus through which the BMC can send and receive IPMI Messages. System software can also access the PCI Management Bus by sending commands to the BMC via the System Interface.
LAN Messaging	O	Ability for the BMC to send and receive IPMI Messaging over LAN
LAN Alerting	O	Ability to send an Alert over the LAN
Serial Messaging	O	Serial messaging is the capability of performing IPMI Messaging over an asynchronous serial connection to the BMC. If Serial Messaging is supported, the following sub-functions apply:
Basic Mode	M	Basic Mode is a type of message framing used for IPMI messaging over a serial connection. Basic Mode support is required if Serial Messaging is supported.
PPP Mode	O	PPP Mode is support for using PPP protocols and framing for IPMI messaging over a serial connection.
Terminal Mode	O	Terminal Mode is a mechanism for IPMI messaging over serial using printable ASCII characters. Terminal mode also supports a limited number of text commands to support legacy ‘text based’ environments.
Direct Connect Mode	M	Direct Connect Mode is support for IPMI messaging over a serial connection without going through a modem. Direct connect mode is mandatory as part of Serial Messaging.
Modem Connect Mode	O	Modem Connect Mode is support for IPMI messaging over a serial connection through a TIA-602-compatible modem, or via modem circuitry that can work with the IPMI commands defined for modem communication.
Bridging Support	O/M	The ability to transfer IPMI request and response messages between two interfaces connected to the BMC. The following support is required if the corresponding interfaces are supported: • serial/modem ←→ IPMB • serial/modem ←→ System Interface • LAN ←→ IPMB • LAN ←→ System Interface Recommended: • serial/modem ←→ PCI Management Bus • LAN ←→ PCI Management Bus Optional: all other combinations, e.g. serial/modem ←→ LAN
Dial Page	O	Ability to perform a numeric page by dialing. Typically accomplished using an external modem.
PPP Alerting	O	Ability for the BMC to connect to a system Platform Event Filtering and Serial Messaging with PPP Mode are required if PPP Alerting is implemented.

Function	M/O	Description
Callback	O	Callback represents the ability for the BMC to be directed to dial up a selected or pre-configured destination to establish an IPMI Messaging session. Callback requires Serial Messaging with Modem Connect Mode.
Basic Mode Callback	M	Required if Callback is supported. BMC uses Basic Mode for IPMI messaging after connecting to specified destination.
PPP Mode Callback	O	BMC uses PPP Mode for IPMI messaging after connecting to specified destination.
CBCP Callback	O	BMC supports Microsoft CBCP (Callback Control Protocol) for callback. PPP Mode and PPP Mode Callback support are required if CBCP Callback is implemented.
Platform Event Filtering (PEF) and Alert Policies	O/M	Ability for BMC to perform a selectable action on an event. This capability is mandatory if paging or alerting is supported. Certain actions within PEF are optional. Refer to the sections on PEF for information. The Alert action and Alert Policies are mandatory if serial/modem or LAN alerting is supported.

4. General Mgmt. Controller Required Functions

All management controllers are required to implement the mandatory IPM Device commands. All other functions are optional. If a function is implemented, such as Event Generation or Sensors, then the mandatory commands for that function must be implemented.

It is highly recommended that all controllers that provide sensors also provide event generation for those sensors. This will eliminate the need for system management software to poll to detect event conditions. It is also highly recommended that all management controllers provide a Primary FRU Inventory device.

5. Message Interface Description

The heart of this specification is the definition of the messages and data formats used for implementing sensors, event messages, Event Generators and Event Receivers, the SDR Repository, and the System Event Log in the platform management architecture. These messages are designed for delivery via a messaging interface with a particular set of characteristics. This section presents the general specification of that interface, and of the messages themselves.

The Message Interface is defined as a 'request/response' interface. That is, a request message is used to initiate an action or set data, and a response message is returned to the Requester . In this document, Request Messages are often referred to as 'commands' or 'requests', and Response Messages as 'responses'.

All messages in this specification share the same common elements as the *payload* to the 'command interpreter' in the logical device that receives the message. The messaging interfaces differ in the framing, physical addressing, and data integrity mechanisms that are used to deliver this payload.

The following are the common components of messages specified in this document:

Network Function (NetFn) — A field that identifies the functional class of the message. The Network Function clusters IPMI commands into different sets. See *Section 5.1, Network Function Codes*, for more information.

Request/Response identifier — A field that unambiguously differentiates Request Messages from Response Messages. In the IPMB Protocol, this identifier is 'merged' with the Network Function code such that 'Even' network function codes identify Request Messages, and 'Odd' network function codes identify Response Messages.

Requester's ID — Information that identifies the source of the Request. This information must be sufficient to allow the Response to be returned to the correct Requester. For example, for the IPMB the Requester's ID consists of the Slave Address and LUN of the Requester device. For a multiple stream system interface the Requester's ID is the 'stream id' for the stream through which the request was issued.

Responder's ID — A field that identifies the Responder to the Request. In Request Messages this field is used to address the Request to the desired Responder, in Response Messages this field is used to assist the Requester in matching up a response with a given request.

Command — The messages specified in this document contain a one-byte command field. Commands are unique within a given Network Function. Command values can range from 00h through FDh. Code FEh is reserved for future extension of the specification, and code FFh is reserved for message interface level error reporting on potential future interfaces.

Data — The Data field carries the additional parameters for a request or a response, if any.

5.1 Network Function Codes

The network layer in the connection header consists of a six-bit field identifying the function to be accessed. The remaining two bits are the LUN field. The LUN field provides further sub-addressing within the node.

The network function is used to cluster commands into functional command sets. In a parsing hierarchy, the LUN field may be thought of as the selector for a particular Network Function handler in the node, and the Network Function may be considered the selector for a particular command set handler within the node.

The following table defines the supported network functions. With the exception of the Application and Firmware Transfer network functions, the commands and responses for a given network function are not node specific. The format and function for standard command sets is specified later in this specification.

Table 5-1, Network Function Codes

Value(s)	Name	Meaning	Description
00, 01	Chassis	Chassis Device Requests and Responses	00h identifies the message as a command/request and 01h as a response, relating to the common chassis control and status functions.
02*, 03*	Bridge	Bridge Requests and Responses	02h (request) or 03h (response) identifies the message as containing data for bridging to the next bus. This data is typically another message, which may also be a bridging message. This function is present only on bridge nodes.
04, 05	Sensor /Event	Sensor and Event Requests and Responses	This functionality can be present on any node. 04h identifies the message as a command/request and 05h as a response, relating to the configuration and transmission of Event Messages and system Sensors.
06, 07	App	Application Requests and Responses	06h identifies the message as an application command/request and 07h a response. The exact format of application messages is implementation-specific for a particular device, with the exception of App messages that are defined by the IPMI specifications. Note that it is possible that new versions of this specification will identify new App commands. To avoid possible conflicts with future versions of this specification, it is highly recommended that the OEM/Group network functions be used for providing 'value added' functions rather than the App network function code.
08, 09	Firmware	Firmware Transfer Requests and Responses	The format of firmware transfer requests and responses matches the format of Application messages. The type and content of firmware transfer messages is defined by the particular device.
0A, 0B	Storage	Non-volatile storage Requests and Responses	This functionality can be present on any node that provides non-volatile storage and retrieval services.
0C, 0D	Transport	Media-specific configuration & control	Requests (0Ch) and responses (0Dh) for IPMI-specified messages that are media-specific configuration and operation, such as configuration of serial and LAN interfaces.
0Eh-2Bh	Reserved	-	reserved (30 Network Functions [15 pairs])
2Ch-2Dh	Group Extension	Non-IPMI group Requests and Responses	The first data byte position in requests and responses under this network function identifies the defining body that specifies command functionality. Software assumes that the command and completion code field positions will hold command and completion code values. The following values are used to identify the defining body: 00h Compact PCI 01h DMTF Pre-OS Working Group ASF Specification all other Reserved When this network function is used, the ID for the defining body occupies the first data byte in a request, and the second data byte (following the completion code) in a response.

Value(s)	Name	Meaning	Description
2Eh-2Fh	OEM/Group	OEM/Non-IPMI group Requests and Response	The first three data bytes of requests and responses under this network function explicitly identify the OEM or non-IPMI group that specifies the command functionality. While the OEM or non-IPMI group defines the functional semantics for the cmd and remaining data fields, the cmd field is required to hold the same value in requests and responses for a given operation in order to be supported under the IPMI message handling and transport mechanisms. When this network function is used, the IANA Enterprise Number for the defining body occupies the first three data bytes in a request, and the first three data bytes following the completion code position in a response.
30h-3Fh	Controller-specific OEM/Group	-	Vendor specific (16 Network Functions [8 pairs]). The Manufacturer ID associated with the controller implementing the command identifies the vendor or group that specifies the command functionality. While the vendor defines the functional semantics for the cmd and data fields, the cmd field is required to hold the same value in requests and responses for a given operation in order for the messages to be supported under the IPMI message handling and transport mechanisms.

* Network Functions that are only utilized in systems that incorporate Bridge nodes.

5.2 Completion Codes

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

Completion Codes work at the 'command' level. They are responses to the interpretation of the command *after* it has been received and validated through the messaging interface. Errors at the 'network' (messaging interface) level are handled with a different error reporting mechanism. For example the SMIC System Interface includes status codes that are separate from the IPMI message data and used to report changes in communication phase or errors in the interface.

Completion Code values are split into 'generic', 'device-specific' (which covers OEM) and 'command-specific' ranges. All commands can return Generic Completion Codes. Commands that complete successfully shall return the 00h, 'Command Completed Normally', Completion Code. Commands that produce error conditions, or return a response that varies from what was specified by the Request parameters for the command, shall return a non-zero Completion Code, as specified in the following table.

Table 5-2, Completion Codes

Code	Definition
	GENERIC COMPLETION CODES 00h, C0h-FFh
00h	Command Completed Normally.
C0h	Node Busy. Command could not be processed because command processing resources are temporarily unavailable.
C1h	Invalid Command. Used to indicate an unrecognized or unsupported command.
C2h	Command invalid for given LUN.
C3h	Timeout while processing command. Response unavailable.
C4h	Out of space. Command could not be completed because of a lack of storage space required to execute the given command operation.
C5h	Reservation Canceled or Invalid Reservation ID.
C6h	Request data truncated.
C7h	Request data length invalid.
C8h	Request data field length limit exceeded.
C9h	Parameter out of range. One or more parameters in the data field of the Request are out of range. This is different from 'Invalid data field' (CCh) code in that it indicates that the erroneous field(s) has a contiguous range of possible values.
CAh	Cannot return number of requested data bytes.
CBh	Requested Sensor, data, or record not present.
CCh	Invalid data field in Request
CDh	Command illegal for specified sensor or record type.
CEh	Command response could not be provided.
CFh	Cannot execute duplicated request. This completion code is for devices which cannot return the response that was returned for the original instance of the request. Such devices should provide separate commands that allow the completion status of the original request to be determined. An Event Receiver does not use this completion code, but returns the 00h completion code in the response to (valid) duplicated requests.
D0h	Command response could not be provided. SDR Repository in update mode.
D1h	Command response could not be provided. Device in firmware update mode.
D2h	Command response could not be provided. BMC initialization or initialization agent in progress.
D3h	Destination unavailable. Cannot deliver request to selected destination. E.g. this code can be returned if a request message is targeted to SMS, but receive message queue reception is disabled for the particular channel.
D4h	Cannot execute command. Insufficient privilege level.
D5h	Cannot execute command. Command, or request parameter(s), not supported in present state.
FFh	Unspecified error.

Code	Definition
	DEVICE-SPECIFIC (OEM) CODES 01h-7Eh
01h-7Eh	Device specific (OEM) completion codes. This range is used for command-specific codes that are also specific for a particular device and version. A-priori knowledge of the device command set is required for interpretation of these codes.
	COMMAND-SPECIFIC CODES 80h-BEh
80h-BEh	Standard command-specific codes. This range is reserved for command-specific completion codes for commands specified in this document.
all other	reserved

5.3 Completion Code Requirements

Completion Codes are provided as an aid in system debugging and error handling. All devices meeting the command specifications of this document shall implement the 00h, 'Command Completed Normally' for the commands specified in this document.

It is mandatory that devices that produce error conditions, or return a response that varies from what was specified by the Request parameters for the command, return a non-zero Completion Code from the preceding table.

In some cases, it is required that a particular completion code be returned for a specified condition. This typically occurs with command-specific completion codes. These cases are documented in the sections describing the particular command or function.

Otherwise, if a device implementation produces a completion condition that matches a Generic or Command-specific completion code for the command, the device shall either return that specific value, or the 'unspecified error' Completion Code, FFh. It is highly desirable that device implementations return an explicit completion code, rather than 'unspecified error', whenever feasible.

In the case that multiple 'non-zero' completion conditions occur simultaneously, the implementation should return whichever completion code the implementer deems to best indicate the condition that the Requester should correct or handle first.

New for IPMI 1.5 Controllers and software that handle IPMI commands: The value C1h (Invalid Command) must be returned for unsupported commands, except when the controller or software is in a mode where general command handling is unavailable. For example, if the controller is in a firmware update mode, it is legal to return D1h (Command response could not be provided, device in firmware update mode) instead of C1h.

5.3.1 Response Field Truncation on non-zero Generic Completion Codes

The responder may, as an implementation option, truncate fields following a non-zero completion code field. Typically, a responder will truncate all fields following a non-zero completion code. If additional fields are returned, however, they should be assumed to have device-specific content.

5.3.2 Summary of Completion Code Use

The following is a summary list of the completion code rules and guidelines.

- A 00h Completion Code must be returned with a normal response to a standard command.
- It is recommended that a 00h Completion Code also be returned for the normal responses to OEM commands.
- A non-zero Completion Code must be returned for an error or atypical response to a standard command.

- It is recommended that a non-zero Completion Code be returned for an error or atypical response to an OEM command.
- The value C1h (Invalid Command) must be returned for unsupported commands, except when in a mode where general command handling is unavailable.
- If the specification calls out that a particular completion code must be returned for a given condition, that code must be returned.
- Otherwise, it is recommended that an implementation return the closest generic completion code for an error condition. If an implementation is resource constrained or the error classification is ambiguous, the FFh (unspecified error) completion code can be returned.
- Device-specific (OEM) completion codes should only be returned when a suitable generic completion code is unavailable. Generic software will treat device-specific completion codes as if they were FFh (unspecified error) completion codes.
- Except for mandatory completion codes, software must not depend on a particular non-zero completion code to be returned for a given error condition, since it is possible that an FFh or device-specific code could be returned instead.
- It is illegal to return a generic or command-specific completion code for a condition that doesn't exist, unless it is being used as part of emulating a device or interface. For example, an implementation might enable the *Master Write-Read* command to be used to access a Private Management Bus interface that is not physically an I^2C bus. The implementation is allowed to return completion codes related to I^2C bus status as part of the emulation.

5.4 Sensor Owner Identification

The definition for the Request/Response identifier, Requester's ID, and Responder's ID are specific to the particular messaging interface that is used. However, the Sensor Data Record and SEL information must contain information to identify the 'owner' of the sensor. For management controllers, a Slave Address and LUN identify the owner of a sensor on the IPMB. For system software, a Software ID identifies the owner of a sensor. These fields are used in Event Messages, where events from management controllers or the IPMB are identified by an eight-bit field where the upper 7-bits are the Slave Address or System Software ID. The least significant bit is a 0 if the value represents a Slave Address and a 1 if the value represents a System Software ID.

The Sensor Number is not part of the Sensor Owner ID, but is a separate field used to identify a particular sensor associated with the Sensor Owner. The combination of Sensor Owner ID and Sensor Number uniquely identify a sensor in the system.

Table 5-3, Sensor Owner ID and Sensor Number Field Definitions

IPMB Sensor Owner ID	System Sensor Owner ID
7:1 Slave Address (7-bits) 0 0b (ID is a slave address)	7:1 System Software ID (7-bits) 0 1b (ID is a Software ID) *See Table 5-4, System Software IDs, below.*
LUN (2-bits)	Sensor Number (8 bits, FFh = reserved)
Sensor Number (8-bits, FFh = reserved)	

5.5 Software IDs (SWIDs)

The following table presents a list of the Sensor Owner IDs for 'system software' sensor owners or IPMI message generators. These values are used when system software issues Event Messages via the system interface, and when

remote console software sends messages to the BMC. For example, if BIOS detects a processor failure, it can generate an Event Message to get the failure event logged. When it formats the Event Message, the BIOS 'System Owner ID' is included in the Event Message. Later, System Management Software can access the System Event Log and tell that the Event Message was generated by BIOS.

For IPMB Messages, the Sensor Owner ID is assumed to be the same as the device that originated the message. Therefore, the Slave Address and LUN of the Event Generator are used. For system-side sensors, it is assumed that the class of software that generates the sensor commands is the 'owner' of the sensor.

Table 5-4, System Software IDs

System Software Type	IDs (7-bit)	bit 0[1]	Resultant 8-bit value[1]
BIOS	00h-0Fh	1b	01h, 03h, 05h, …1Fh
SMI Handler	10h-1Fh	1b	21h, 23h, 25h, … 3Fh
System Management Software	20h-2Fh	1b	41h, 43h, 45h, … 6Fh
OEM	30h-3Fh	1b	61h, 63h, 65h, … 7Fh
Remote Console software 1-7	40h-46h	1b	81h, 83h, 87h, … 8Dh
Terminal Mode Remote Console software	47h	1b	8Fh
reserved	remaining	1b	-

1. The System Software ID is often used in an 8-bit field where the least-significant bit is a 1b to indicate that the field holds a Software ID rather than a slave address. One example of this occurs in the first byte of the Generator ID field in an event message. The last column in the above table illustrates how the 7-bit Software ID appears in such a 1-byte field.

5.6 Isolation from Message Content

The SEL, SDR, and Event commands are designed so that the 'devices' that implement those command sets are generally isolated from the content of the SEL Entry, Sensor Data Record, and Event Message contents. That is, the Event Receiver device receives and routes Event Messages, but does not interpret them. Similarly, the SEL and SDR devices retrieve and store log entries and Sensor Data Records, respectively, without interpreting their contents.

6. IPMI Messaging Interfaces

This section introduces the common characteristics of the messaging interfaces to the BMC and between the BMC and system software. As mentioned earlier, there are three System Interface implementations specified for the BMC: SMIC, KCS, and BT. The BMC can also be communicated with through additional interfaces such as the IPMB, ICMB, LAN, and Serial/Modem interfaces. Information specific to the operation and usage of a particular interface is given in later sections.

6.1 Terminology

The ICMB, LAN, and Serial/Modem interfaces are typically used to communicate with management software on another system. The remote software that is used to communicate with the BMC is referred to as the *remote console*. Although the word 'console' is used, the remote software may or may not provide a user interface or require user interaction.

Local software running on the managed system and using the System Interface to the BMC will generally be referred to as *system management software* or *SMS.* Unless otherwise indicated, the direction of communications is given with respect to the BMC. E.g. transmitted, outbound, or outgoing messages are issued from the BMC. Received, inbound, or incoming messages are accepted by the BMC. Other terminology used for IPMI Messaging will be introduced as it is used in the following sections.

6.2 Channel Model

IPMI uses a 'channel model' for directing communication between different interfaces in the BMC. *Channels* serve as the means for identifying the medium for a messaging interface, and for configuring user information and passwords, message authentication, access modes and privilege limits associated with that interface.

Each channel has its own set of configuration parameters for user information and channel privilege limits. This allows different sets of user names and passwords and different levels and types of authentication to be used on different channels. IPMI Messaging and Alerting can also be independently enabled or disabled for an entire channel on a per channel basis.

Channels share commands related to authentication, access, and configuration. These commands are independent from the type of communication medium. This reduces the amount of medium-specific information that software needs to deal with, and simplifies task such as bridging IPMI messages between different media.

6.3 Channel Numbers

Each interface has a *channel number* that is used when configuring the channel and for routing messages between channels. Only the channel number assignments for the primary IPMB and the System Interface are fixed, the assignment of other channel numbers can vary on a per-platform basis. Software uses a *Get Channel Info* command to determine what types of channels are available and what channel number assignments are used on a given platform. The following table describes the assignment and use of the channel numbers:

Table 6-1, Channel Number Assignments

Channel Number	Type/Protocol	Description
0	Primary IPMB	Channel 0 is assigned for communication with the primary IPMB. IPMB protocols are used for IPMI messages. Refer to [IPMB] for more information.
1-7	Implementation-specific	Channels 1-7 can be assigned to different types types of communication media and protocols for IPMI messages (e.g. IPMB, LAN, ICMB, etc.), based on the system implementation. For IPMI 1.5, 'Channel Protocol Type' and 'Channel Medium Type' numbers identify the channel's protocol and medium, respectively. Software can use the *Get Channel Info* command to retrieve this information.
8h-Dh	-	Reserved
Eh	Present I/F	The value Eh is used as a way to identify the current channel that the command is being received from. For example, if software wants to know what channel number it's presently communicating over, it can find out by issuing a *Get Channel Info* command for channel E.
Fh	System Interface	Channel 'F' is assigned for routing messages to the system interface.

6.4 Channel Protocol Type

The protocol used for transferring IPMI messages on a given channel is identified using a channel protocol type number. In earlier versions of the specification, this also implied the particular medium for the channel. The *Channel Medium Type* number is now used to explicitly indicate the class of the media. Both these values are used in the *Get Channel Info* command.

Sensor Data Record 14h - BMC Message Channel Info is superceded by the Get Channel Info command. New implementations should use the *Get Channel Info* command instead.

Table 6-2, Channel Protocol Type Numbers

Channel Protocol (5-bits)	Name	Description
00h	n/a	reserved
01h	IPMB-1.0	Used for IPMB, serial/modem Basic Mode, and LAN
02h	ICMB-1.0	ICMB v1.0 - See *Section 8, ICMB Interface*.
03h	reserved	reserved
04h	IPMI-SMBus	IPMI on SMBus 1.x - 2.x [1] Request = [rsSA, Netfn(even)/rsLUN, 00h, rqSA, rqSeq/rqLUN, CMD[2], <data>, PEC] Response = [rqSA or rqSWID, NetFn(odd)/rqLUN, 00h, rsSA or rsSWID, rqSeq/rsLUN, CMD, completion code, <data>, PEC]
05h	KCS	KCS System Interface Format - See *Section 9,Keyboard Controller Style* (KCS) Interface
06h	SMIC	SMIC System Interface Format - See *Section 10, SMIC Interface*
07h	BT-10	BT System Interface Format, IPMI v1.0 - See *Section 11, Block Transfer (BT) Interface*
08h	BT-15	BT System Interface Format, IPMI v1.5 - See *Section 11, Block Transfer (BT) Interface*
09h	TMode	Terminal Mode - See *Section 13.7, Terminal Mode*
1Ch-1Fh	n/a	OEM Protocol 1 through 4, respectively
all other		reserved

1. Note that the IPMI format is intentionally illegal with respect to the SMBus specification protocols in order to provide a way for a management controller to unambiguously differentiate IPMI messages from SMBus transactions. This enables a management controller to support both SMBus and IPMI protocols without concern that they would overlap. The PEC (packet error code) is an 8-bit CRC calculated per the SMBus 2.0 specification. This format makes it simple to use the same hardware or firmware routines for data integrity checking of both IPMI and SMBus messages.)
2. Note that certain network functions, such as OEM/Group, require additional standard fields within the <data> portion of a message.

6.5 Channel Medium Type

The Channel Medium Type number is a seven-bit value that identifies the general class of medium that is being used for the channel.

Table 6-3, Channel Medium Type Numbers

Channel Type	Description
0	reserved
1	IPMB (I^2C)
2	ICMB v1.0
3	ICMB v0.9
4	802.3 LAN
5	Asynch. Serial/Modem (RS-232)
6	Other LAN
7	PCI SMBus
8	SMBus v1.0/1.1
9	SMBus v2.0
Ah	reserved for USB 1.x
Bh	reserved for USB 2.x
Ch	System Interface (KCS, SMIC, or BT)
60h-7Fh	OEM
all other	reserved

6.6 Channel Access Modes

Session-based channels can be configured to provide IPMI Messaging access only when the system is in certain states. This allows the system user to configure various levels of security and remotely accessible features. The access modes are summarized in the *Table 6-4, Channel Access Modes*. Commands allow the power-up default (non-volatile) Access Mode for a channel to be configured, and allow the Access Mode setting to be changed dynamically. Channel Access Modes are configured using the *Set Channel Access* command. The *Set Channel Access* command is also used to enable or disable Alerting for the entire channel.

Support for any given access mode is implementation specific. It is expected that most implementations will support Disabled and Always Available, and that serial/modem channels will also support the Shared access mode.

Table 6-4, Channel Access Modes

Pre-boot Only	The channel is only available out-of-band while the machine is powered-off and during POST until the boot process is initiated. This option is primarily used with Serial Port Sharing where it may be desirable to ensure that the BMC does not take control of the serial port during OS run-time. The BMC will not claim the port once the port has been switched over to the system using the 'force mux' option in the *Set Serial/Modem Mux* command, unless the system becomes powered down or is reset. As a consequence, run-time software must rely on mechanisms such as the IPMI Watchdog Timer to power down or reset the system in order to enable communication to the BMC under failure conditions. There is a Modem Ring Time parameter in the serial/modem channels that configures the amount of time that the BMC waits for RING before directing the modem to connect. This parameter can be used to enable the BIOS to 'answer the phone' instead of the BMC. See *Table 13-2, Serial Port Sharing* Access Characteristics for more information. LAN Channels do not typically allow setting Pre-boot Access Mode. If it is provided, BIOS should disable the channel at the end of POST (start of boot) by using the *Set Channel Access* command to set the channel to 'disabled' using the volatile setting. The Pre-boot Only setting does not affect serial/modem alerting. If alerting is enabled and software does not handle the event, the BMC will take control of the port/channel for the time it takes to deliver the alert. Alerting can be enabled or disabled for an entire channel via the *Set Channel Access* command.
Always Available	The channel is dedicated to communication with the BMC and is available during all system states (powered-down, powered-up, pre-boot, sleep, run-time, etc.) For IPMI LAN channels, this means that RMCP packets are handled by the BMC. For serial/modem channels on systems that support serial port sharing, the port can still be switched over to the system, however the BMC will always 'answer the phone' and respond to escape sequences and packets that activate the port. The BIOS will typically disable software access to the serial port when it sees the BMC configured for Always Available mode. This is done to prevent any possible confusion between auto-answer applications running on the OS and the BMC's answering of the phone.
Shared	The channel can be shared between system software and the BMC. Shared Mode is typically only used when there is a need to switch the communication resource between system software and the BMC because the system and BMC cannot readily interleave their traffic on the medium, as is the case with Serial Port Sharing. For IPMI LAN Channels, Shared Mode means that the implementation allows system software to receive and respond to RMCP packets. However, this does not prevent the BMC from handling IPMI RMCP packets and RMCP Ping/Pong. If software wanted exclusive access to RMCP Packets, it would need to temporarily disable IPMI messaging by setting the volatile setting of the access mode to 'disabled'. Note that if system software failed, a system reset (e.g. watchdog reset) or power down would be required to restore LAN communication with the BMC. For serial/modem channels that support Serial Port Sharing, the system BIOS will typically leave the baseboard serial port available for software use when it sees this mode set. This allows system software to use the port and any external modem for 'outgoing' traffic, while the BMC can still 'answer the phone' for incoming calls. Thus, in shared mode, the mux will be set to 'system' whenever the BMC is not in the process of answering a call or handling or establishing an IPMI messaging session. There is a Modem Ring Time parameter in the serial/modem channels that configures the amount of time that the BMC waits for RING before directing the modem to connect. This parameter can be used to enable 'auto answer' OS applications, while providing a way to connect to the BMC if a failure

	prevents the run-time application from answering the phone. If the Modem Ring Time is set to a non-zero wait time, the BMC will leave the mux set to the system until the Modem Ring Time expires, at which time the BMC can answer the phone. If the Modem Ring Time is set to a zero wait time, the BMC will take the mux and attempt to answer the phone as soon as it detects an incoming call. See *Table 13-2, Serial Port Sharing Access Characteristics* for more information.
Disabled	The channel is disabled from being used to communicate with the BMC. The Disabled setting does not affect alerting. Alerting is separately enabled or disabled via a separate field in the *Set Channel Access* command.

6.7 Logical Channels

From the IPMI Messaging point-of-view, a party that bridges a message from one channel to another only is mainly concerned that it gets the correct response from the BMC. Often, it doesn't matter to remote console or system software whether the target channel and devices are physically implemented or not. For example, a BMC could implement a logical IPMB where the BMC would respond to messaging commands as if there was a physical IPMB with other management controllers on it. An implementation might elect to do this for several reasons. One reason would be that the board vendor wanted to use an alternative bus for interconnecting the management functions within their board set. Another possibility is that a logical IPMB could provide a way to organize add-on functions to the BMC, such as embedding a logical ICMB Bridge controller.

6.8 Channel Privilege Levels

Channel *privilege limits* determine the maximum privilege that a user can have on a given channel. One channel can be configured to allow users to have up to Administrator level privilege, while another channel may be restricted to allow no higher than User level. The privilege level limits take precedence over the privilege level capabilities assigned per user.

Channels can be configured to operate with a particular maximum Privilege Level. Privilege levels tell the BMC which commands are allowed to be executed via the channel. *Table 6-5, Channel Privilege Levels*, lists the currently defined privilege levels. The *Set Channel Access* command is used to set the maximum privilege level limit for a channel. The *Set Session Privilege Level* Command is used to request the ability to perform operations at a particular privilege level. The *Set Session Privilege Level* command can only be used to set privilege levels that are less than or equal to the privilege level limit for the entire channel, regardless of the privilege level of the user.

Table 6-5, Channel Privilege Levels

Callback	This may be considered the lowest privilege level. Only commands necessary to support initiating a Callback are allowed. *Appendix G - Command Assignments*, provides a list of the commands that are executable when operating at Callback Level.
User	Only 'benign' commands are allowed. These are primarily commands that read data structures and retrieve status. Commands that can be used to alter BMC configuration, write data to the BMC or other management controllers, or perform system actions such as resets, power on/off, and watchdog activation are disallowed. *Appendix G - Command Assignments*, provides a list of the commands that require operating at User level or higher.
Operator	All BMC commands are allowed, except for configuration commands that can change the behavior of the out-of-band interfaces. For example, Operator privilege does not allow the capability to disable individual channels, or change user access privileges. *Appendix G - Command Assignments*, provides a list of the commands that require operating at Operator level or higher.
Administrator	All BMC commands are allowed, including configuration commands. An Adminstrator can even execute configuration commands that would disable the channel that the Administrator is communicating over. *Appendix G - Command Assignments*, provides a list of the commands that require operating at Administrator level.

6.9 Users & Password Support

The term *user* is used in this specification to refer to a collection of data that identifies a password (key) for establishing an authenticated session, and the privileges associated with that password. For configuration purposes, the sets of user information are organized and accessed according to a numeric *User ID.* When activating a session, user information is looked up using a text *username*.

User access can be enabled on a *per channel* basis. Thus, different channels can have different sets of users enabled.

If desired, a username on one channel can be associated with a different password than the same username on a different channel. When a session is activated, the BMC will scan the usernames sequentially starting with User ID 1 and will look for the first user that has a matching username and has access enabled for the given channel. Thus, having different passwords for a given username requires configuring multiple user entries - one for each different password that is to be used for a particular set of channels.

The specification allows a number of different implementations for supporting users on a channel. The following lists the minimum requirements:

- All authenticated channels are required to support at least one user (User ID 1).
- Usernames may be fixed or configurable, or a combination of both, at the choice of the implementation.
- If an implementation supports only one user with a fixed user name, then the fixed user name must be null (all zeros).
- Support for configuring user passwords for all User IDs is required.
- Support for setting per-user privilege limits is optional. If the *Set User Access* command is not supported, the privilege limits for the *channel* are used for all users.

6.9.1 'Anonymous Login' Convention

The IPMI convention for enabling an 'anonymous' login is to configure the entry for User ID 1 with a null username (all zero's) and a null password (all zero's). Applications may then present this to the user as an anonymous login and configuration option, knowing what username and password to use if the BMC allows 'anonymous' logins. The reason for doing this via User ID 1 is to simplify the task of enabling the BMC to report whether anonymous login is enabled or not.

6.9.2 Anonymous Login Status

The *Get Channel Authentication Capabilities* command includes a 'Anonymous Login Status' field. This field indicates to a remote console application whether User ID 1 is presently configured with a null username and null password. In addition, a bit is provided that indicates whether there are also non-null usernames enabled for the channel, or whether User ID 1 holds a null username, but a non-null password.

Together, these bits can be used to guide a remote application in presenting connection options to a user. For example, if a system only has Anonymous Login enabled, the application could immediately connect without prompting the user, or use that information to enable an 'anonymous login' button in the user interface. If a system has a null username but non-null password, the application could put up just a password dialog box. Lastly, if the system indicates it has non-null usernames with non-null passwords, the application may put up a dialog box prompting for both a user name and password.

6.10 System Interface Messaging

The following sections describe how messaging works across the system interface to the BMC. Later sections go into detail on message formats and register interfaces for the different physical implementation options for the system interface.

6.10.1 BMC Channels and Receive Message Queue

Messaging between system software and the other management busses, such as the IPMB, is accomplished using channels and a *Receive Message Queue*. A channel is a path through the BMC that allows messages to be sent between the system interface and a given bus or message transport. The Receive Message Queue is used to hold message data for system software until system software can collect it. All channels share the Receive Message Queue for transferring messages to system management software. The Receive Message Queue data contains channel, session, and IPMI addressing information that allows system software to identify the source of the message, and to format a message back to the source if necessary.

System management software is responsible for emptying the Receive Message Queue whenever it has data in it. *Messages are rejected if the Receive Message Queue gets full.* It is recommended that the Receive Message Queue have at least two 'slots' for each channel. The Receive Message Queue is a logical concept. An implementation may choose to implement it as an actual queue, or could implement separate internal buffers for each channel. It is recommended that the implementation attempt to leave a slot open for each channel that does not presently have a message in the queue. This helps prevent 'lock out' by having the queue fill with just messages from one interface.

The BMC itself can, if necessary, use the Receive Message Queue and Messaging Channels to send asynchronous messages to system management software. The recommended mechanism for accomplishing this is to define a unique channel with a protocol type of 'System'. To send an asynchronous message to system software, the BMC would place a message from that channel directly into the Receive Message Queue in 'System' format. System software would be able to respond back to the BMC using a *Send Message* command for that channel.

6.10.2 Event Message Buffer

[Optional] - The Event Message Buffer holds Event Request Messages that have been internally generated by the BMC, and Event Messages that have been received by the BMC from the IPMB or other channel. The Event Message Buffer does *not* hold event messages that have been generated from system software.

The Event Message Buffer holds all 16-bytes of the Event Message as it would be stored in the System Event Log (see *Table 26-1, SEL Event Records*). For IPMI v1.5, the Event Message Buffer does not get overwritten if a new event comes in before system software can empty the buffer. The BMC does clear the buffer when the BMC is first powered up and whenever the system becomes powered up or is hard reset. A BMC implementation can support generating a system interrupt when the Event Message Buffer gets filled.

Some implementations will elect to generate an SMI to allow the creation of an SMI Handler that takes additional actions on Event Messages. If Event Message Buffer interrupts do not generate SMI, or are not enabled (or not implemented), SMS can use this as a mechanism for examining Event Messages as they are received. System software must check the status of SMI use before assuming that the Event Message resource is available. This can be accomplished by using the *Get Channel Info* command to determine if the interrupt assignment for the Event Message Buffer is set to SMI.

> *Note: SMM Messaging and the implementation of SMIs is OPTIONAL. Since SMI operation and functions are proprietary and not described nor required in this specification, support via the IPMI interfaces is being deprecated. New implementations should avoid using the IPMI support for SMI.*

6.11 IPMI Sessions

Authenticated IPMI communication to the BMC is accomplished by establishing a session. Once established, a session is identified by a Session ID. The Session ID may be thought of as a handle that identifies a connection between a given remote user and the BMC, using either the LAN or Serial/Modem connection.

The specification supports having multiple active sessions established with the BMC. It is recommended that a BMC implementation support at least four simultaneous sessions. This number is shared between the LAN and Serial/Modem interfaces.

The specification also allows a given endpoint (identified by an IP address) on the LAN to open more than one session to a BMC. The capability is allowed to allow a single system to serve as a proxy to provide BMC LAN sessions for other systems. It is not intended for one system to use this provision to open multiple sessions to the BMC for that system's sole use.

An IPMI messaging connection to the BMC fits one of three classifications, session-less, single-session, or multi-session.

6.11.1 Session-less Connections

A session-less connection is unauthenticated. There is no 'user login' required for performing IPMI messaging. The System Interface and IPMB are examples of session-less connections.

6.11.2 Single-session Connections

A single-session connection has a user authentication phase that precedes IPMI messaging. This is accomplished using the *Get Session Challenge* and *Activate Session* commands. A single-session connection is intended for a physically secure link. Therefore, individual packets are not signed. The serial/modem Basic Mode is an example of a single-session connection.

6.11.3 Multi-session Connections

A multi-session connection has user authentication and supports multiple interleaved sessions (multiple users). The multi-session connection is specified to support communication on a shared medium, such as LAN, where there may be a mix of IPMI and non-IPMI traffic. In order to support multiple sessions, and protect against attempts to circumvent authentication (such as replay attacks), multi-session packets have a *session header* in addition to the IPMI message. The session header carries information to identify the particular session, as well as other fields such as session sequence numbers and authentication type fields. The LAN and PPP Mode connections are examples of multi-session IPMI messaging connections.

6.11.4 Per-Message and User Level Authentication Disables

Typically, each packet in a multi-session connection is authenticated (with the exception of the packets for certain 'pre-session' commands such as *Get Channel Authentication Capabilities*, and *Get Session Challenge*.) In some cases however, the connection medium is considered to be trusted even though multiple user sessions are allowed. Once a session has been activated, the computational overhead of authenticating each packet may not be necessary.

Thus, there are two options to enable performance improvements in environments where the link is considered to be secure. The options are to disable 'Per-Message Authentication', and to disable 'User Level Authentication'. If Per-Message Authentication is disabled, the only packets that are required to be authenticated are the ones for the *Activate Session* request and response. Once the session is activated, the remaining packets will be accepted with the Authentication Type set to NONE. Since the Authentication Code (signature) is not

provided in the packet when the Authentication Type is NONE, this enables a performance improvement in two ways: fewer bytes are transmitted, and the authentication algorithm doesn't need to be run.

In many cases, there is little concern about whether User Level commands are authenticated, since the User privilege allows status to be retrieved, but cannot be used to cause actions such as platform resets, or change platform configuration. Thus, an option is provided to disable authentication just for User Level commands. If User Level Authentication is disabled, then User Level messages will be accepted that have the Authentication Type set to NONE.

The BMC will always verify any authenticated packets (Authentication Type not NONE) that it receives, regardless of whether Per-Message Authentication and/or User Level Authentication is disabled. Authenticated packets will be silently discarded if the signature (AuthCode) is invalid, or the Authentication Type does not match the authentication type that was negotiated when the session was activated. This is necessary to allow remote console software to deliver authenticated messages to the Receive Message Queue via the Send Message command.

Both the Per-Message Authentication and User Level Authentication disable options are configured via the *Set Channel Access* command.

6.11.5 Link Authentication

Sometimes connections offer authentication protocols that are applied as part of establishing the communication link to the BMC. For example, PPP supports authentication protocols such as PAP and CHAP that are part of link establishment.

Link Authentication is a global characteristic associated with the connection mode for the channel. Link Authentication is enabled/disabled via the *serial/modem configuration parameters.* When Link Authentication is enabled, it is necessary to identify one or more users that will serve as the source of the username (peer ID) and password information for the link. This is accomplished by setting an 'Enable User for Link Authentication' bit in the *Set Channel Access* command.

For physically secure connections, these 'Link Authentication' protocols may be all that's considered needed to authenticate the user. Thus, the BMC supports enabling Link Authentication for PPP using common PPP authentication algorithms. If Link Authentication is enabled, the Per-Message Authentication Disable, and User Level Authentication Disable options may be used to improve performance.

6.11.6 Summary of Connection Characteristics

The following table summarizes the key characteristics that differentiate session-less, single-session, and multi-session connections:

Table 6-6, Session-less , Single-session and Multi-session Characteristics

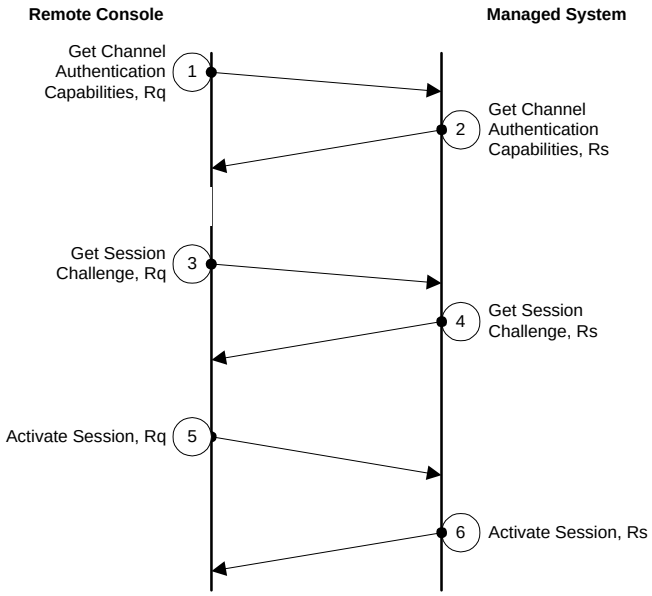
	Multi-Session	Single-Session	Session-less	Session Header	Authenticated Access	Per Message Authentication Disable	User Level Authentication Disable	Link Authentication
LAN	X			X	X	X	X	
Serial/Modem:								
PPP Mode	X			X	X	X	X	X
Basic Mode		X			X			
Terminal Mode		X			X[1]			
IPMB			X					
ICMB			X					
PCI Management Bus			X					

1. Terminal mode only supports ‘straight password’ authentication

6.11.7 Session Activation and IPMI Challenge-Response

A session must be activated before general IPMI messaging can occur. The mechanism for accomplishing this is via a set of IPMI commands that are used to perform an "IPMI Challenge-Response". The session activation process involves three IPMI commands: *Get Channel Authentication Capabilities*, *Get Session Challenge*, and *Activate Session.* Of these three commands, the *Get Channel Authentication Capabilities* and *Get Session Challenge* command must be executable before the session is set up. Therefore, these commands can be thought of as always being 'unauthenticated'. The *Activate Session* command is the first, and in some cases only, authenticated command for a session.

Figure 6-1, Session Activation



Referring to *Figure 6-1, Session Activation*, the following presents the general steps for activating a session:

1. The Remote Console issues a *Get Channel Authentication Capabilities* request to the BMC.

2. The BMC returns a *Get Channel Authentication Capabilities* response that contains which authentication types (authentication algorithms) it supports.

3. The Remote Console sends a *Get Session Challenge* request to the BMC. The command selects which of the BMC-supported authentication types the Remote Console would like to use, and a username that selects which set of user information should be used for the session. This is the only place where the username is used during the process.

4. The BMC looks up the user information associated with the username. If the user is found and allowed access via the given channel, the BMC returns a *Get Session Challenge* response that includes a randomly generated Challenge String and a temporary Session ID. The BMC keeps track of the username associated with the Session ID so that it can use the Session ID to look up the user's information in the next step. In some algorithms, the BMC will store challenge string, or a seed that was used to generate the challenge, for later lookup as well.

5. The Remote Console then issues an *Activate Session* request. The request contains the temporary Session ID plus the authentication information based on the type of authentication that was selected. For example, a LAN packet would typically include a signature using an authentication algorithm run

on elements such as the challenge string, user password/key, IPMI message fields, Session ID, etc. while a serial/modem connection may only pass a simple clear-text password in the activate session data. The authentication format for different authentication types is specified in the description of the *Activate Session* command. For multi-session connections, the starting Outbound session sequence that the BMC is to use when sending packets to the remote console is also passed in the request. (Session sequence numbers are explained in the next section.)

6. The BMC uses the temporary Session ID to look up the information for the user that was identified in the *Get Session Challenge* request. The BMC looks up the user's password/key data, and potentially other data such as a stored copy of the earlier challenge string, and uses it to verify that the packet signature or password is correct. If so, the BMC issues an *Activate Session* response that provides the Session ID to use for the remainder of the session. For multi-session connections, the Activate Session response is itself authenticated (signed). The BMC will also deliver the starting Inbound session sequence that the Remote Console is to use when sending packets to the BMC.

From this point, whether individual packets for the session are authenticated or not is based on settings such as the Per User Authentication and User Level Authentication parameters. Refer back to *6.11.4, Per-Message and User Level Authentication Disables*.

6.11.8 Session Sequence Numbers

The session sequence number is a 32-bit, unsigned, value. The session sequence number is *not* used for matching IPMI requests and responses. The IPMI Sequence (Seq) field is used for that purpose. The session sequence number is solely for protecting against replay attacks.

There are two Session Sequence Numbers: the *Inbound* Session Sequence Number and the *Outbound* Session Sequence Numbers. The inbound and outbound directions are defined with respect to the BMC. Inbound messages are from the remote console to the BMC, while outbound messages are from the BMC to the remote console.

Inbound messages use the inbound session sequence number, while outbound messages use the outbound session sequence number.

6.11.9 Session Sequence Number Generation

Session sequence numbers are generated on a per-session basis. The inbound and outbound sequence numbers are updated and tracked independently. The BMC and the remote console independently select the starting session sequence number for the messages they receive.

The remote console sets the starting values for the outbound session sequence number when it sends the first *Activate Session* command for an authenticated session.

The *Activate Session* response is the first authenticated outbound (BMC to remote console) message. This response message uses the initial outbound session sequence number value that the remote console delivered in the prior *Activate Session* command request. The BMC must increment the outbound session sequence number by one (1) for each subsequent outbound message from the BMC.

The first authenticated inbound message uses the initial inbound (remote console to BMC) session sequence number value that was returned by the BMC in the response to the *Activate Session* command. The remote console must increment the inbound session sequence number by one (1) for each subsequent message it sends to the BMC.

6.11.10 Inbound Session Sequence Number Tracking and Handling

Session sequence numbers are tracked on a per-session basis. At a minimum, the BMC is required to track that the inbound sequence number is increasing, and to silently discard the packet if the sequence number is **eight**

counts or more from than the last value received. (An implementation is allowed to contain a proprietary configuration option that enables a larger sequence number difference, as long as the standard of +eight can be restored.)

An implementation can elect to terminate the session if it receives a number of sequence numbers that are more than eight counts from the last value received.

Valid packets (packets with good data integrity checks and signature) to a given session that have the same inbound sequence number as an earlier packet are considered to be duplicate packets and are silently discarded (even if the message content is different).

6.11.11 Out-of-order Packet Handling

In order to avoid closing a session because a packet was received out-of-order, the BMC must implement one of two options:

Option 1: Advancing eight-count (or greater) window. Recommended. Track which packets have been received that have sequence numbers up to eight counts *less* than the highest last received sequence number, tracking which of the prior eight sequence numbers have been received. Also accept packet with sequence numbers that are up to eight counts greater than the last received sequence number, and set that number as the new value for the highest sequence number received. This option is illustrated in *Appendix A - Previous Sequence Number Tracking*.

Option 2: Drop any packets with sequence numbers that are lower than the last valid value received. While simpler than option 1, this option is not recommended except for resource-constrained implementations due to the fact that any out-of-order packets will require the remote console to timeout and retransmit.

Sequence number wrap-around must be taken into account for both options. When a sequence number advances from FFFF_FFFFh to 0000_0000h, the value FFFF_FFFFh represents the lesser sequence number.

6.11.12 Outbound Session Sequence Number Tracking and Handling

The remote console is required to handle outbound session sequence number tracking in the same manner as the BMC handles the inbound session sequence number, except that Option 2 (above) should not be used as a means of handling out-of-order packets.

6.11.13 Session Inactivity Timeouts

A session is automatically closed if no new, valid, message has been received for the session within the specified interval since the last message. The session must be re-authenticated to be restored. A remote console can optionally use the *Activate Session* command to keep a session open during periods of inactivity.

Note that only an active session will keep the Session Inactivity Timeout from expiring. IPMI message activity that occurs outside of an active session has no effect. This is to prevent someone from keeping a phone connection indefinitely while trying to guess different passwords to activate a session.

The BMC only monitors for inactivity while the connection is switched over to the BMC. Note that closing a session is not always the same as hanging up a modem connection. Serial/modem sessions are also automatically closed when the connection is switched over to the system, but the phone connection remains active. The BMC only terminates the phone connection if a session is closed due to an inactivity timeout while the serial connection is routed to the BMC.

The timeout and tolerance values are specified for the management controller (BMC) that will timeout and close the session. System software should take this tolerance into account, plus any additional delays due to media transmission times, etc.

An implementation can provide an option to allow the timeout to be configurable via a parameter in the configuration parameters for the given channel type.

Table 6-7, Default Session Inactivity Timeout Intervals

Session Type	Default Expiration Interval	Tolerance	Notes
LAN	60 seconds	+/- 3 seconds	
Direct Connect Mode Serial	60 seconds	+/- 3 seconds	
Modem Mode Serial	60 seconds	+/- 3 seconds	The Inactivity Timeout Interval starts whenever a connection is established with, or switched to, the BMC. The Phone connection gets terminated if inactivity timeout occurs while serial connection is routed to BMC.

6.11.13.1 Avoiding 'Slot Stealing'

It is highly recommended that an implementation provide a mechanism for protecting against someone accidentally or maliciously 'claiming' all the session slots and subsequently locking out access to the BMC. For example, this could occur by an errant program repeatedly issuing *Get Session Challenge* commands without successfully activating a session - causing all available resources for tracking pending sessions to be used up.

One possible solution is to use an LRU algorithm that drops the session ID for the oldest session ID that has a pending 'Activate Session'. That way, the only way to 'permanently' use up slots is by activating and maintaining sessions for all session slots. A minor refinement may be to provide a few seconds of delay on returning the response to the *Get Session Challenge* in order to give opportunity for a well-behaved application to get a Challenge and return an *Activate Session* command before the errant software re-issued another *Get Session Challenge*. (This is only an improvement for errant applications that wait for the response to the *Get Session Challenge* before issuing the request again.)

6.11.14 Additional Session Specifications and Characteristics

- At least four simultaneous sessions should be supported on a given channel.
- By default, sessions are automatically closed if no valid activity is detected within the Session Inactivity Timeout Interval**,** or if the connection or link is terminated. Valid activity is defined as the receipt of a valid IPMI message for that session while the connection is routed to the BMC.
- At least four pending bridged requests should be supported for each bridged interface that requires the BMC to track pending responses. See *6.12, BMC Message Bridging* for more information.
- The typical BMC is expected to allow only a small number of simultaneous open sessions (on the order of four to eight). Thus, remote console applications should avoid activating multiple sessions whenever possible, in order to allow other remote consoles to also get access.
- The *Activate Session* command will return a completion code indicating whether the request was rejected because BMC is presently busy with other open sessions.
- The specification allows multiple sessions to be activated from a single IP address. The primary reason for allowing multiple sessions is to allow a system to serve as a proxy agent that provides BMC access for remote consoles that connect to it instead of directly to the BMC.
- Multiple sessions are *not* intended to be used to support access by multiple applications behind an IP address. If multiple applications require access to the BMC on a given system, a single driver or 'middle-ware' should coordinate that access and use a single session, if possible. The IPMI Software ID and the

IPMI sequence number are two fields that a shared driver can used to identify and route messages to and from a given application.

- There is a 1:1 relationship between a user name and a session. I.e. different usernames cannot share the same session. However, multiple sessions can be activated using the same username.
- All sessions start off at User Level privilege. It is necessary to issue a *Set Session Privilege* to raise the operating privilege level before commands that required higher privilege can be executed. The maximum operating privilege for a session is determined by Privilege Limits that are set both for the user and for the overall channel. The more restrictive setting of the User Privilege Limit and the Channel Privilege Limit sets the maximum operating privilege available for a session.
- An Operator can optionally use the *Get Channel Info* and *Get Session Info* commands to retrieve the address of parties with open sessions and their present privilege level. This is to allow a remote console to coordinate with another remote console that already has an active session. This can be used to allow software to coordinate access to the system. For example, one system
- An Administrator can force sessions on any channel to be terminated.

6.12 BMC Message Bridging

BMC Message Bridging provides a mechanism for routing IPMI Messages between different media. Bridging is only specified for delivering messages between different channels; i.e. it is not specified for delivering messages between two sessions on the same channel.

In IPMI 1.0, bridging was primarily specified just for providing access between SMS (System Interface) and the IPMB. With IPMI 1.5, these mechanisms have been extended to support delivering IPMI messages between active connections / sessions on any IPMI Messaging media connected to the BMC.

There are three mechanisms for bridging messages between different media connected to the BMC, depending on what the target of the message is:

- **BMC LUN 10b** is used for delivering messages to the System Interface. The BMC automatically routes any messages it receives via LUN 10b to the Receive Message Queue.
- ***Send Message* command from System Interface** is used for delivering messages to other channels, such as the IPMB. The messages appear on the channel as if they've come from BMC LUN 10b. Thus, if the message is a request message, the response will go to BMC LUN 10b and the BMC will automatically place the response into the Receive Message Queue for retrieval. System software is responsible for matching the response up with the original request, thus the 'No Tracking' setting in the *Send Message* command is used.
- ***Send Message* command with response tracking.** This format of *Send Message* command is used with response tracking for bridging request messages to all other channels except when the System Interface is the source or destination of the message.

The following sections provide additional information on the operation and use of these bridging mechanisms.

6.12.1 BMC LUN 10b Routing

Because messages to SMS are always routed to the Receive Message Queue, the *Send Message* command is not typically used to send messages to SMS. Instead, messages to SMS are delivered via the BMC SMS LUN, 10b. The BMC automatically reformats and places any messages that are addressed to LUN 10b into the Receive Message Queue for SMS to retrieve using the *Get Message* command.

Thus, sending a request to SMS just requires formatting the command so that it is addressed to BMC LUN 10b. SMS can retrieve the request from the Receive Message Queue, extract the originator's address and channel info, and then use the *Send Message* command to deliver a response.

The BMC does not track requests and responses for messages to system software because the Receive Message Queue provides the channel and session information necessary to format the *Send Message* command to deliver the response. Similarly, system software is capable of tracking the channel and session information it used when generating a request. Thus, the 'No Tracking' option is used for *Send Message* commands from system software.

The responder then delivers its response message to BMC LUN 10b and the response gets routed to the Receive Message Queue. Conversely, if a channel wants to deliver a message to SMS, it sends the request message to BMC LUN 10b, and later SMS uses a Send Message command to return the response from BMC LUN 10b.

6.12.2 Send Message Command From System Interface

The operation of the *Send Message* command when issued via the *System Interface* is different than when the *Send Message* command is issued from other interfaces. This is because the IPMI System Interfaces were specified as always returning an immediate command response. In order to avoid tying up the System Interface waiting for a bridged response, a response to the *Send Message* command is returned as soon as the request is bridged to the target channel. This response only indicates that the Send Message command was executed. It is not the response to the bridged request.

Later, the response to the bridged request is received by the BMC and routed into the Receive Message Queue and it is retrieved using a *Get Message* command. For example, here are the typical steps involved in delivering a request from the System Interface to a device on the IPMB, and receiving a corresponding response:

1. System software formats a *Send Message* request that encapsulates information for the request to be placed on IPMB. The requester's LUN in the data is set to 10b so when the response comes back, the BMC will place it in the Receive Message Queue. The encapsulated request is also given a sequence number by the system software. System software will use this number later, along with other fields, to match up the Receive Message Queue data with the original request.

2. System software delivers the *Send Message* request to the BMC via the System Interface.

3. The BMC returns an 'OK' response to the Send Message command, indicating that it has received the request and delivered it to the IPMB.

4. Sometime later, the target IPMB device delivers a response to the request. The response is sent back to the same requester's LUN that was used in the request, 10b. The BMC routes message data received on 10b to the receive message queue, and also tags it with information such as the channel number that the message was received from.

5. System software detects that there is data in the Receive Message Queue. This is either done by polling for messages by periodically checking the SMS_ATN bit, or for interrupt driven implementations, getting an interrupt when SMS_ATN becomes set. Software then uses the *Get Message Flags* command to discern whether the SMS_ATN condition was from getting data into the Receive Message Queue or some other event.

6. System software then issues a *Get Message* request. The response returns a message from the queue. If the data is for a response, software then checks the message fields, such as sequence number, channel number, CMD, etc., to see if the response matches up with an earlier request. In this example, software would be looking for a response to the request it had bridged onto IPMB. If the Receive Message Queue holds a request for system software, it processes it accordingly.

7. If software has not received a response by the timeout intervals specified for IPMB, it can retry the request. Also note that IPMB sequence numbers generally expire after **5 Seconds**. This number comes from the sequence number expiration interval on IPMB. Software can generally discard requests that are more than 5 seconds old and re-use their sequence numbers.

If the target channel uses sessions, the *Send Message* command data will require a *Session Handle* value to select which session on the channel the message will be sent to. Software can use the *Get Channel Info* and *Get*

Channel Sessions commands to determine what channels are present and to obtain the Session Handle for a given session.

6.12.3 Send Message Command with Response Tracking

The *Send Message* command is used primarily to direct the BMC to act as a proxy that translates a message from one IPMI messaging protocol to another. The BMC formats the data for the target channel type and protocol and delivers it to the selected medium.

Media such as the IPMB do not include channel number and session information as part of their addressing information. As a result, request messages from another channel must be delivered as if they originated from the BMC itself.

If the bridged message is a request, it is necessary for the BMC to hold onto certain data, such as originating channel and session information, so that when the response message comes back it can reformat the response and forward it back to the originator of the request. The primary way the BMC accomplishes this is by assigning a unique sequence number to each request that it generates, and saving a set of information in a 'Pending Bridged Response' table that is later used to reformat and route a response back to the originator of the request.

The sequence number returned in the response is then used to look up who generated the original response, plus the saved formatting and addressing information. The BMC then reformats and delivers the response to the original requester and deletes the request from its list of 'pending responses'. The *Send Message* command includes a parameter that directs the BMC to save translation information for and track outstanding request messages for the purpose of routing the response back to the originator of the Send Message command.

Note that, with the exception of messages to SMS, when the Send Message command is used to deliver a message to a given medium the message appears to have been originated by the BMC. This means that a controller on the IPMB can't generically distinguish a bridged request from SMS from a bridged request from LAN.

Table 6-8, Message Bridging Mechanism by Source and Destination

Message Type and direction	Delivery Mechanism	BMC tracks pending responses
Request or Response from System Interface to any other channel	Send Message	no
Request or Response to System Interface from any other channel	BMC LUN 10b	no
Request from any channel except System Interface to IPMB	Send Message	yes
Response from IPMB to any channel except System Interface	BMC LUN 00b	yes
Request from any channel (except System Interface) to PCI Management Bus	Send Message	yes
Request from PCI Management bus to any channel except System Interface	BMC LUN 00b	yes
Request from Serial to LAN	Send Message	yes
Response LAN to Serial	BMC LUN 00b	yes
Request from LAN to Serial	Send Message	yes
Response from Serial to LAN	BMC LUN 00b	yes

6.12.4 Bridged Request Example

The example illustrates a *Send Message* command from LAN being used to deliver a request to IPMB.

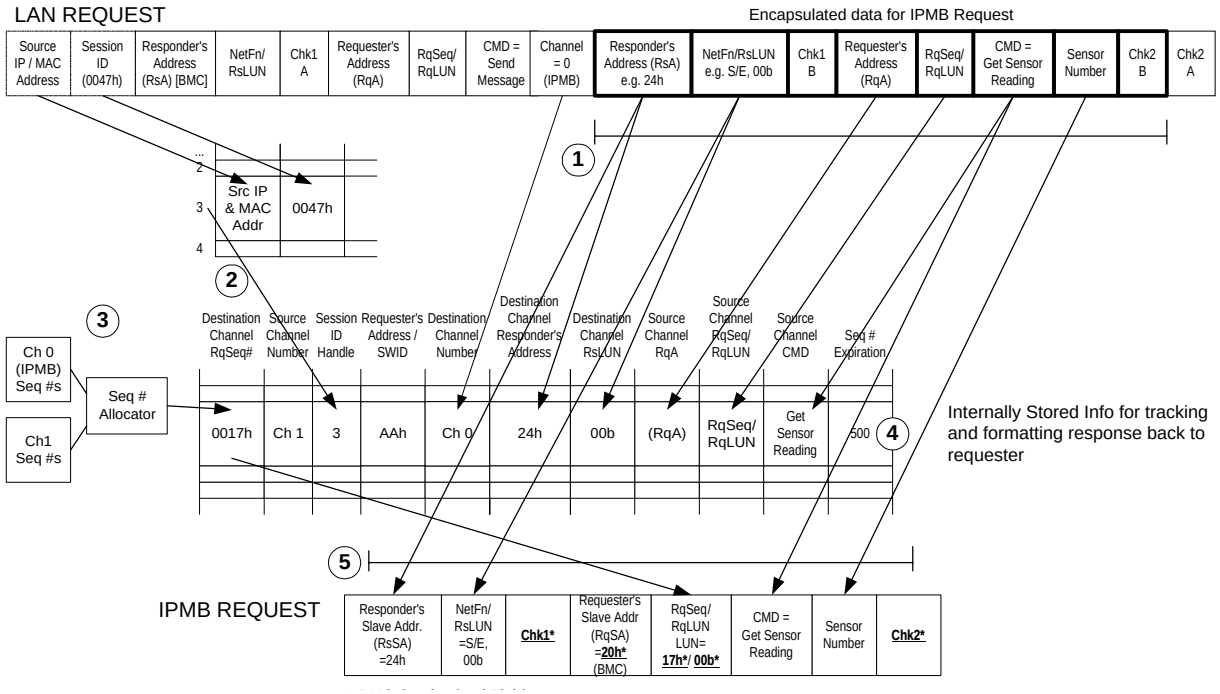
Bridged requests to the IPMB can come from several different channels: LAN, serial/modem, and the ICMB. The BMC uses the sequence number that it places on the bridged request to identify which channel and to which address on that channel the response is to go back to. It is therefore important for the BMC to ensure that unique sequence numbers are used for pending requests from the different channels. It is also important that sequence numbers are unique for successive requests to a given responder. One way to manage sequence numbers to the

IPMB is to track sequence numbers on a per responder basis. This can be kept in a table of 'Pending Bridged Response' info.

In order to get the response back to the LAN, the IPMB response must return the same sequence number that was passed in the request. (This is a basic rule of IPMI Messaging, so there's nothing special about that requirement.) The management controller uses the sequence number to look up the channel type specific addressing, sequence number, and security information that it stored when the request was forwarded. For example, if the channel type is 'LAN' then the response message must be formatted up in an RMCP/UDP packet with the IP address of the requester, the sequence number passed in the original request, the appropriate security 'key' information, etc.

The following figure and steps present an example high-level design for handling a bridged request. Note that the example shows information that is generated and stored, but it does not show any particular code module that would perform that operation. That is, the choice of which functions are centralized, which are in a 'LAN' module, and which are in an 'IPMB' module (or whether you even have such modularity) is left to the implementer.

Figure 6-2, LAN to IPMB Bridged Request Example



1. When the BMC receives the *Send Message* command with the 'Bridged Request' parameter bit set, it checks for an available entry in a *Pending Bridged Response* table and copies parameters from the request to be bridged. When the response comes back, these parameters will be used to validate that the response matches the earlier request and to reformat the response for the originating channel. The bold outlined boxes represent parameters and data in the *Send Message* command that will ultimately be copied to the resulting request on the target channel (the IPMB in this example).

2. Any channel session information necessary to get the response back to the original requester will also need to be recorded. In this example, the BMC maintains a separate table of session information for the LAN channel. An offset into that table is used as a 'handle' for identifying the session information associated with the request. This handle is used in the *Pending Bridged Response* table in lieu of copying all the session information. Note that with such an implementation, it is important to remember details such as invalidating and freeing any bridge table entry associated with that session if the session should get closed while responses are pending.

3. In this example, the BMC has a separate 'Sequence Number Allocator' routine that ensures that sequence numbers used in bridged requests are kept unique for a given channel. This is done so when the response comes back, the sequence number can be used to look up the corresponding request info entries from the *Pending Bridged Response* table.

4. Responses have a five second 'sequence number expiration' interval. If a response is not received by the expiration interval, the corresponding entry in the *Pending Bridged Response* entry is deleted and the *sequence* number associated with the request can be reused. The Seq # Expiration column in this example represents a possible implementation where the Seq # Expiration value is decremented nominally once every 10 ms. The entry is considered to be free when the number hits 0. Thus, in this example the Seq # Expiration field could be used both for tracking sequence number expiration as well as a mechanism for marking whether a table entry is available or not.

5. The BMC takes the indicated values and uses them to construct the bridged request. The request is a combination of field values copied from the original Send Message command and values generated by the BMC. The BMC generated values are shown with a **bold underlined** typeface with an asterisk.

6.13 Message Size & Private Bus Transaction Size Requirements

The following table summarizes the message size and transaction size requirements for the various interfaces to the BMC and IPMI Management controllers. The IPMI message sizes include any IPMI-level addressing and data integrity information required for the interface. For example, the IPMB Message lengths include the requester and responder addressing information, sequence number, and checksums. The message sizes do not include counts for additional encapsulation, data escaping, or framing used to transport the IPMI message on the given media.

The IPMB standard overall message length for 'non-bridging' messages is specified as 32 bytes, maximum, including slave address. This sets the upper limit for typical IPMI messages. With the exception of messages used for bridging messages to other busses or interfaces (e.g. Master Write-Read and Send Message) IPMI messages should be designed to fit within this 32-byte maximum. In order to support bridging, the Master Write-Read and Send Message commands are allowed to exceed the 32-byte maximum transaction on IPMB.

Refer to *Appendix D - Determining Message Size Requirements* for information on how these values were derived.

Table 6-9, IPMI Message and IPMB / Private Bus Transaction Size Requirements

Interface	Requirement Description	
KCS/SMIC Input	Required:	**40 bytes** IPMI Message, minimum
KCS/SMIC Output	Required:	**38 bytes** IPMI Message, minimum
BT Input	Required:	**42 bytes** IPMI Message, minimum, (including BT Interface 'length' byte). The BT interface has length and Seq fields in addition to the fields used by the KCS and SMIC interfaces. This adds two bytes to the message size support requirements.
BT Output	Required:	**40 bytes** IPMI Message, minimum, (including BT Interface 'length' byte)
IPMB Input	Required: Recommended:	32 bytes IPMI Message, minimum (including slave address) **36 bytes** bus transaction, minimum (including slave address), to support an OEM option to allow the BMC to be the target of an SMBus 2.0 Block-Write with PEC.
IPMB Output	Required:	**36 bytes** bus transaction, minimum (including slave address) to allow the BMC to be able to issue access slave devices that use the SMBus 2.0 Block-Write with PEC protocol. Note that the IPMB standard message length is shorter than the SMBus 2.0 message. *The IPMB standard message length is specified as 32 bytes, maximum, including slave address.*
SMBus 2.0 Input	Required:	**36 bytes** bus transaction, minimum (including slave address) to allow the BMC to be target of an SMBus 2.0 Block-Write with PEC protocol transaction.
SMBus 2.0 Output	Required:	**36 bytes** (including slave address) to allow the BMC to generate an SMBus 2.0 Block-Write with PEC transaction.
Private Bus Input	Recommended:	**34 bytes** bus transaction, minimum, if the Private Bus is implemented as a physical I^2C or SMBus. The 34 bytes supports accessing slave devices that use the SMBus 2.0 Block-Read protocol. (The count excludes any slave address, since for this type of transaction the slave address is output by the management controller, rather than being an input to the management controller.)

Private Bus Output	Required:	**23 bytes** This is only required when a controller indicates that it has a Private Bus that includes FRU SEEPROMs that are accessible via the Master Write-Read command must support a *Master Write-Read* command equivalent to the largest *Master Write-Read* command that could be delivered as a 32-byte IPMB message. Otherwise, the Private Bus is only required to support the transaction size required for the private or OEM devices that are used in the particular implementation. An implementation will typically implement a private bus using an actual I^2C or SMBus connection. However, the private bus implementation could be 'virtual' - where the management controller responds to the *Master Write-Read* command as if a physical private bus were present. For a physical private bus implementation, a 32-byte *Master Write-Read* command in IPMB format results in one byte of slave address and 22 bytes of write data going to the private bus.
	Recommended:	**36 bytes** bus transaction, minimum (including slave address). A private bus that truly implements a physical I^2C or SMBus interface should support system management software access to slave devices that use the SMBus 2.0 Block Write protocol. This means supporting a *Master Write-Read* command over the system interface that can be used to perform a full, 36-byte SMBus 2.0 Block-Write protocol transaction.
LAN/PPP Input	Required:	**45 bytes** IPMI Message content, minimum. IPMI LAN and PPP interfaces must accept an RMCP Packet containing an IPMI Message that would allow the remote console to submit a *Master Write-Read* message to perform an SMBus 2.0 Block-Write protocol transaction. Since the LAN interface uses a message format that follows the IPMB message format, there are additional bytes for source and destination addressing, sequence number, and checksums.
LAN/PPP Output	Required:	**42 bytes** IPMI Message content, minimum. IPMI LAN and PPP interface must support delivering an RMCP Packet containing an IPMI Message that would allow the BMC to return the response to a *Master Write-Read* message that returns data from an SMBus 2.0 Block-Read protocol transaction.

7. IPMB Interface

7.1 IPMB Access via Master Write-Read command

When an IPMB is implemented in the system, the BMC serves as a controller to give system software access to the IPMB. The IPMB allows non-intelligent devices as well as management controllers on the bus. To support this operation, the BMC provides the *Master Write-Read* command via its interface with system software. The *Master Write-Read* command provides low-level access to non-intelligent devices on the IPMB, such as FRU SEEPROMs.

The *Master Write-Read* command provides a subset of the possible I^2C and SMBus operations that covers most I^2C/SMBus-compatible devices.

In addition to supporting non-intelligent devices on the IPMB, the *Master Write-Read* command also provides access to non-intelligent devices on Private Busses behind management controllers. The main purpose of this is to support FRU SEEPROMs on Private Busses.

7.2 BMC IPMB LUNs

A BMC supports several LUNs (Logical Units) to which messages can be sent via the IPMB interface. These LUNs are used to identify different sub-addresses within the BMC that messages can be sent to.

Table 7-1, BMC IPMB LUNs

LUN	Short Description	Long Description
00b	BMC commands and Event Request Messages	Event Request Messages received on this LUN are routed to the Event Receiver function in the BMC, and automatically logged if SEL logging is enabled .
01b	OEM LUN 1	OEM - reserved for BMC implementer / system integrator definition.
10b	SMS Message LUN (Intended for messages to System Management Software)	Messages received on this LUN are routed to the Receive Message Queue and retrieved using a *Read Message* command. The SMS_Avail flag is set whenever the Receive Message Queue has valid contents.
11b	OEM LUN 2	OEM - reserved for BMC implementer / system integrator definition.

7.3 Sending Messages to IPMB from System Software

System Management Software (SMS) can use the BMC for sending and receiving IPMB messages. Both IPMB request and response messages can be sent and received using this mechanism. Therefore, not only can system software send requests to the IPMB and receive responses from the IPMB, it is also possible for system software to receive requests *from* the IPMB to send back IPMB responses.

System software sends messages to the IPMB through the system interface using the BMC as an IPMB controller. This is accomplished by using the *Send Message* command to write the message to the IPMB (channel 0). The BMC does not place any restrictions on the type or content of the IPMB message being sent. System management software can send any IPMB request or response message it desires provided that the message meets the maximum length requirements of the *Send Message* command.

System Management Software is responsible for providing all fields for the IPMB message, including Requester and Responder Slave addresses and checksums. The following figures show an example using the *Send Message* command to send a *Set Event Receiver* command to an IPMB device at slave address 52h, LUN 00b, via the

system interface (see *Table 23-2, Set Event Receiver*). The example command sets the Event Receiver address to 20h = BMC.

The heavy bordered fields show the bytes for the IPMB message carried in the *Send Message* command. The requester's LUN field (rqLUN) is set to 10b (BMC SMS LUN). This directs the responder to send the response to the *Set Event Receiver* command to the BMC's Receive Message Queue.

Figure 7-1, IPMB Request sent using Send Message Command

NetFn (06h = App request)	LUN (00b)	Command (Send Message)	Channel (00h)	
Slave address for write (52h = rsSA)	NetFn/rsLUN (04h / 00b = Sensor/Event, LUN 00b)			check 1 (9Eh)
rqSA (20h = BMC)	rqSeq/rqLUN (000001b / 10b, 10b = SMS LUN)		Cmd (00h = Set Event Receiver)	
event receiver slave address (20h = BMC)	event receiver LUN (00h)	check 2 (BAh)		

Figure 7-2, Send Message Command Response

NetFn (07h = App response)	LUN (00b)	Command (Send Message)	Completion Code (00h)

Note that the response is for the *Send Message* command, *not* for the *Set Event Receiver* command. The response to the *Set Event Receiver* command will be returned later in the Receive Message Queue. System software uses the *Get Message* command to read messages from the Receive Message Queue. System software keeps track of any outstanding responses and matches responses up with corresponding requests as they come in. System software must also keep track of the protocol assigned to the particular channel in order to interpret the response to the *Get Message* command.

7.4 Sending IPMB Messages to System Software

It is possible for devices on the IPMB to autonomously send messages to system management software via the BMC. IPMB messages that are addressed to the SMS LUN (10b) in the BMC are placed into the Receive Message Queue. The contents can then be retrieved using the *Get Message* command. System management can then interpret the message and use the *Send Message* command to return a response.

The BMC does not place any restrictions on the type of content of the IPMB message being received, as long as it is properly formatted, is addressed to the SMS LUN, and meets the maximum length requirements of the *Get Message* command.

The BMC sets the corresponding 'ATN' flag in the system interface when a message is received into the Receive Message Queue. System software must poll for the 'ATN' flag, or receive an interrupt to determine that when a message is available.

Event Messages can also be made directly available to system software via the optional Event Message Buffer and retrieved using the *Read Event Message Buffer* command.

In the example shown in the preceding section, a *Set Event Receiver* command was sent out on the IPMB using the *Send Message* command. In the IPMB command, the requester's slave address (rqSA) was set to 20h (BMC), and the requester's LUN (rqLUN) set to 10b (SMS LUN). This means that the response will be sent to the SMS Message Buffer in the BMC.

The IPMB response to a *Set Event Receiver* command consists of just a Completion Code byte in the data portion of the IPMB message. Assuming a Completion Code of 00h = OK, the Receive Message Queue will eventually wind up a response message with the following contents:

Figure 7-3, Response for Set Event Receiver in Receive Message Queue

NetFn (05h = Sensor/Event Response)		rqLUN (10b = SMS LUN)	Check 1 (CAh)	
rsSA (52h)	rqSeq/rsLUN (000001b / 00b)	Cmd (00h = Set Event Receiver)	Completion Code (00h = OK)	Check 2 (AAh)

Note that this is the entire IPMB response message, with the leading slave address stripped off (the leading slave address does not need to be stored, since its known to be the BMC slave address = 20h).

The response to the *Get Message* command would then look like the following. The heavy border fields show the data portion of the response that came from the Receive Message Queue.

Figure 7-4, Get Message Command Response

NetFn (07h = App response)	LUN (00b)	Command (Get Message)	Completion Code (00h)	Channel Number (00h)
NetFn (05h = Sensor/Event Response)		rqLUN (10b = SMS LUN)	Check 1 (CAh)	
rsSA (52h)	rqSeq/rsLUN (000001b / 00b)	Cmd (00h = Set Event Receiver)	Completion Code (00h = OK)	Check 2 (AAh)

7.5 Testing for Event Message Buffer Support

System software must test for Event Message Buffer support. If software issues a *Get BMC Global Enables* command, and finds the buffer enabled, it can assume the controller supports the buffer. Otherwise, it must test by attempting to enable the Event Message buffer.

If the BMC does not support the desired buffer, it shall return an Invalid Data Field (CCh) error completion code when an attempt is made to enable the respective buffer using the *Set BMC Global Enables* command. An error completion code shall also be returned if an attempt is made to enable Event Message Buffer interrupts when that option is not supported.

8. ICMB Interface

The ICMB Specification (see [ICMB]) describes the interfaces for implementing access via an ICMB Bridge Controller. ICMB was specified so that an ICMB Bridge controller could be added to an existing IPMI Implementation that contained an IPMB.

In some implementations, the BMC can serve as the ICMB Bridge Controller. There are two ways to implement the interface for such a controller:

a. Implement a virtual ICMB Bridge Controller within the BMC.
b. (IPMI v1.5 only) Implement the ICMB Bridge commands directly as BMC commands, but use IPMI v1.5 channels and the *Send Message* command to replace the ICMB *Bridge Request* command.

8.1 Virtual ICMB Bridge Device

In this implementation, the ICMB Bridge Device functionality appears to system software as if there was a separate ICMB Bridge Controller on a physical IPMB. Instead, the BMC implements the ICMB Bridge Device functions internally on a 'virtual IPMB'. This option can provide backward-compatibility with software that was designed to work with a non-integrated ICMB Bridge Device.

The BMC reports that the Chassis Bridge Device is not part of the BMC by returning an address in the *Get Chassis Capabilities* that is different than the BMC address (20h). This indicates to software that it need to access the Bridge Device function by using *Send Message* commands to deliver messages to the Bridge Device via the primary IPMB. The BMC then monitors the *Send Message* command for messages directed to the Bridge Device address. When the BMC sees *Send Message* commands to the Bridge Device address, it handles them internally instead of routing them out to a physical IPMB. Responses from the virtual Bridge Device are placed into the *Receive Message Queue* as if they were received from the IPMB.

It is optional for the BMC to provide ICMB access from the IPMB for this implementation. If such support is desired, there are two implementation options. The first option is for the BMC to respond to two slave addresses (the BMC address and the Bridge Device address). The second option is for the BMC to report the BMC address as the Bridge Device address whenever the *Get Chassis Capabilities* command is received from IPMB, and implement the bridge commands directly when accessed via the IPMB.

8.2 ICMB Bridge Commands in BMC using Channels

In this implementation, the BMC directly responds to the commands for the Chassis Bridge Device. The BMC reports its own address in the *Get Chassis Capabilities* command. This tells software that it does not need to use the *Send Message* command to encapsulate messages in order to access the Chassis Bridge function itself. This also tells software that is must use the *Send Message* command instead of the *Bridge Request* command.

8.2.1 ICMB Bridging from System Interface to Remote IPMB using Channels

The behavior with the *Send Message* command is somewhat different than the operation of the *Bridge Request* command implemented by a separated bridge controller on the IPMB. When the *Bridge Request* command was used to access the ICMB, the response to that command would hold the response from the ICMB.

From the System Interface, bridging with the *Send Message* command is a multi-step operation: First you issue the *Send Message* command with the data to be sent to the ICMB. You then get a response to the *Send Message* command indicating that the data was successfully bridged onto the ICMB. This response does not contain the response data from the ICMB. Later (assuming that a request was bridged) the device on ICMB will respond and the response data will appear in the *Receive Message Queue* (if the System Interface was the source of the

original *Send Message* command). The software can then use a *Get Message* command to retrieve the response message data.

The following tables show the KCS formats of the *Send Message* command request and response for bridging a request to a device on a remote IPMB and the later corresponding Receive Message Queue contents for the response from the remote device.

Table 8-1, System Interface Request For Delivering Remote IPMB Request via ICMB

NetFn/RsLUN	App (even=Rq) / BMC LUN = 00b
CMD	*Send Message*
Data 1	Channel Number = ICMB, track request = 1b
Data 2	rqSeq = sequence number selected by system software / 00b
Data 3:4	rmtBrXA
Data 5	*Bridge Request* CMD
Data 6	rsSA for remote IPMB device
Data 7	netFn / rsLUN for remote IPMB device
Data 8	CMD for remote IPMB device
Data 9:N	Data for remote IPMB device
Checksum	Checksum for *Send Message* Command

Table 8-2, Send Message Response

NetFn/RsLUN	App (odd=Rs) / BMC LUN = 00b
CMD	*Send Message*
Data 1	Completion Code for *Send Message* command
Checksum	Checksum for *Send Message* Command

Table 8-2, Get Message Response Data for Remote IPMB Request Delivered via ICMB

NetFn/RsLUN	App (odd=Rs) / BMC LUN = 00b
CMD	*Get Message*
Data 1	Completion Code for *Get Message* command
Data 2	Channel Number = ICMB
Data 3	rqSeq = Sequence number from original request / 00b
Data 4	rmtBr Completion Code
Data 5	remote IPMB completion code
Data 6:N	remote IPMB data
Checksum	Checksum for *Send Message* Command

8.2.2 ICMB Bridging from Local IPMB to Remote IPMB using Channels

From the IPMB, bridging with the *Send Message* command operates in a manner similar to the way it operates over the system interface. The main difference being that the device that originated the request later receives an asynchronous response message that appears as if the BMC is responding directly to the remote IPMB command.

Note that the same rqSeq is used both in the response to the *Send Message* command and in the asynchronous response from the BMC.

Bridging with this approach introduces a five-byte overhead on the request, and a 0-byte overhead on the response.

Table 8-3, IPMB Request For Delivering Remote IPMB Request via ICMB

RsSA	20h (BMC)
NetFn/RsLUN	App (even=Rq) / 00b (BMC LUN)
RqSA	Address of local IPMB device issuing the request
RqSeq/RqLUN	Sequence number selected by IPMB device issuing the request / LUN of IPMB device issuing the request
CMD	*Send Message*
Data 1	Channel Number = ICMB, track request = 1b
Data 2:3	rmtBrXA
Data 4	*Bridge Request* CMD (Tells BMC to deliver this to ICMB a message to be bridged to a remote IPMB)
Data 5	rsSA for remote IPMB device
Data 6	netFn / rsLUN for remote IPMB device
Data 7	*Remote CMD* (CMD for remote IPMB device)
Data 8:N	*Remote Data* (data for remote IPMB device)
Checksum	Checksum for *Send Message* Command

Table 8-4, Send Message Response

RqSA	Address of local IPMB device that issued the original request
NetFn/RqLUN	App (odd-Rs) / LUN of device that issued the original request
RsSA	20h (BMC)
RqSeq / RsLUN	Sequence number from original request / 00b (BMC LUN)
NetFn/RsLUN	App (odd=Rs) / BMC LUN = 00b
CMD	*Send Message*
Data 1	Completion Code for *Send Message* command
Checksum	Checksum for *Send Message* Command

Table 8-5, IPMB Response For Remote IPMB Request Delivered via ICMB

RqSA	Address of local IPMB device that issued the original request
NetFn/RqLUN	App (odd-Rs) / LUN of device that issued the original request
RsSA	20h (BMC)
RqSeq / RsLUN	Sequence number from original *Send Message* request / 00b (BMC LUN)
NetFn/RsLUN	App (odd=Rs) / BMC LUN = 00b
CMD	*Remote CMD*
Data 1	*Remote CMD* Completion code
Data 4:N	*Remote CMD* data
Checksum	Checksum for *Send Message* Command

9. Keyboard Controller Style (KCS) Interface

This section describes the Keyboard Controller Style (KCS) Interface. The KCS interface is one of the supported BMC to SMS interfaces. The KCS interface is specified solely for SMS messages. SMM messages between the BMC and an SMI Handler will typically require a separate interface, though the KCS interface is designed so that system software can detect if a transaction was interrupted. Any BMC-to-SMI Handler communication via the KCS interface is implementation specific and is not covered by this specification.

The KCS Interface is designed to support polled operation. Implementations can optionally provide an interrupt driven from the OBF flag, but this must not prevent driver software from the using the interface in a polled manner. This allows software to default to polled operation. It also allows software to use the KCS interface in a polled mode until it determines the type of interrupt support. Methods for assigning and enabling such an interrupt are outside the scope of this specification.

9.1 KCS Interface/BMC LUNs

LUN 00b is typically used for all messages to the BMC through the KCS Interface. LUN 10b is reserved for Receive Message Queue use and should not be used for sending commands to the BMC. Note that messages encapsulated in a *Send Message* command can use any LUN in the encapsulated portion.

9.2 KCS Interface-BMC Request Message Format

Request Messages are sent to the BMC from system software using a *write transfer* through the KCS Interface. The message bytes are organized according to the following format specification:

Figure 9-1, KCS Interface/BMC Request Message Format

Byte 1	Byte 2	Byte 3:N
NetFn/LUN	Cmd	Data

Where:

LUN Logical Unit Number. This is a sub-address that allows messages to be routed to different 'logical units' that reside behind the same physical interface. The LUN field occupies the least significant two bits of the first message byte.

NetFn Network Function code. This provides the first level of functional routing for messages received by the BMC via the KCS Interface. The NetFn field occupies the most significant six bits of the first message byte. Even NetFn values are used for requests to the BMC, and odd NetFn values are returned in responses from the BMC.

Cmd Command code. This message byte specifies the operation that is to be executed under the specified Network Function.

Data Zero or more bytes of data, as required by the given command. The general convention is to pass data LS-byte first, but check the individual command specifications to be sure.

9.3 BMC-KCS Interface Response Message Format

Response Messages are *read transfers* from the BMC to system software via the KCS Interface. Note that the BMC only returns responses via the KCS Interface when Data needs to be returned. The message bytes are organized according to the following format specification:

Figure 9-2, KCS Interface/BMC Response Message Format

Byte 1	Byte 2	Byte 3	Byte 4:N
NetFn/LUN	Cmd	Completion Code	Data

Where:

LUN Logical Unit Number. This is a return of the LUN that was passed in the Request Message.

NetFn Network Function. This is a return of the NetFn code that was passed in the Request Message. Except that an odd NetFn value is returned.

Cmd Command. This is a return of the Cmd code that was passed in the Request Message.

Completion Code The Completion Code indicates whether the request completed successfully or not.

Data Zero or more bytes of data. The BMC always returns a response to acknowledge the request, regardless of whether data is returned or not.

9.4 Logging Events from System Software via KCS Interface

The KCS Interface can be used for sending Event Messages from system software to the BMC Event Receiver. The following figures show the format for KCS Interface Event Request and corresponding Event Response messages. Note that only Event Request Messages to the BMC via the KCS Interface have a Software ID field. This is so the Software ID can be saved in the logged event.

Figure 9-3, KCS Interface Event Request Message Format

NetFn (04h = Sensor/Event Request)			LUN (00b)	Command (02h = Platform Event)		Software ID (Gen ID) 7-bits	1	
EvMRev	Sensor Type	Sensor #	Event Dir	Event Type	Event Data 1		Event Data 2	Event Data 3

Shading designates fields that are not stored in the event record.

Figure 9-4, KCS Interface Event Response Message Format

NetFn (**05h** = Sensor/Event Response)	00	Command (02h = Platform Event)	Completion Code

9.5 KCS Interface Registers

The KCS Interface defines a set of I/O mapped communication registers. The bit definitions, and operation of these registers follows that used in the Intel 8742 Universal Peripheral Interface microcontroller. The term 'Keyboard Controller Style' reflects the fact that the 8742 interface is used as the system keyboard controller interface in PC architecture computer systems.

The specification of the KCS Interface registers is given solely with respect to the 'system software side' view of the interface in system I/O space. The functional behavior of the management controller to support the KCS

Interface registers is specified, but the physical implementation of the interface and the organization of the interface from the management controller side is implementation dependent and is beyond the scope of this specification.

On the system side, the registers are mapped to system I/O space and consists of four byte-wide registers.

- Status Register - provides flags and status bits for use in various defined operations.
- Command Register - provides port into which 'Write Control Codes' may be written.
- Data_In - provides a port into which data bytes and 'Read Control Codes' may be written.
- Data_Out - provides a port from which data bytes may be read.

The default system base address for an I/O mapped KCS SMS Interface is **CA2h.** Refer to *Appendix C - Locating IPMI System Interfaces via SM BIOS Tables* for information on using SM BIOS tables for describing optional interrupt usage, memory mapped registers, 32-bit and 16-byte aligned registers, and alternative KCS interface addresses. Software can assume the KCS interface registers are I/O mapped and byte aligned at the default address unless other information is provided.

Figure 9-5, KCS Interface Registers

	7	6	5	4	3	2	1	0	I/O address
Status (ro)	S1	S0	OEM2	OEM1	C/D#	SMS_ATN	IBF	OBF	base+1
Command (wo)									base+1
Data_Out (ro)									base+0
Data_In (wo)									base+0

Reserved bits must be written as '0' and ignored during reads. Software should not assume that reserved bits return a constant value.

9.6 KCS Interface Control Codes

Control codes are used for 'framing' message data transferred across the KCS Interface. Control Codes are used to:

- Identify the first and last bytes of a packet.
- Identify when an error/abort has occurred.
- Request additional data bytes.

9.7 Status Register

System software always initiates a transfer. If the BMC has a message for SMS, it can request attention by setting the SMS_ATN bit in the status register. System software then detects the flag and initiates the transfer.

Other bits in the status register are used to arbitrate access to the command and data registers between the BMC and system software and to indicate the "state" (write, read, error, or idle) of the current transaction. The following tables summarize the functions of the Status Register bits.

Table 9-1, KCS Interface Status Register Bits

Bit	Name	Description	R/W[1]
7	S1	State bit 1. Bits 7 & 6 are used to indicate the current state of the KCS Interface. Host Software should examine these bits to verify that it's in sync with the BMC. See below for more detail.	R/O
6	S0	State bit 0. See bit 7.	R/O
5	OEM2	OEM - reserved for BMC implementer / system integrator definition.	R/O
4	OEM1	OEM - reserved for BMC implementer / system integrator definition.	R/O
3	C/D#	Specifies whether the last write was to the Command register or the Data_In register (1=command, 0=data). Set by hardware to indicate whether last write from the system software side was to Command or Data_In register.	R/O
2	SMS_ATN	Set to 1 when the BMC has one or more messages in the Receive Message Queue, or when a watchdog timer pre-timeout, or event message buffer full condition exists[2]. OEMs may also elect to set this flag is one of the OEM 1, 2, or 3 flags from the *Get Message Flags* command becomes set. This bit is related to indicating when the BMC is the source of a system interrupt. Refer to sections *9.12, KCS Communication and Non-communication Interrupts, 9.13, Physical Interrupt Line Sharing*, and *9.14, Additional Specifications for the KCS interface* for additional information on the use and requirements for the SMS_ATN bit.	R/O
1	IBF	Automatically set to 1 when either the associated Command or Data_In register has been written by system-side software.	R/O
0	OBF	Set to 1 when the associated Data_Out register has been written by the BMC.	R/O

1. R/W direction is with respect to the system side of the interface. Reads move data from the BMC to system software, writes move data from system software to the BMC.
2. The event message buffer full condition contributes to SMS_ATN only if the event buffer full condition is intended to be handled by system management software. Otherwise, the event message buffer full condition should not contribute to SMS_ATN. For interrupt driven interfaces, the condition is required to contribute to SMS_ATN if the event message buffer full condition generates the same interrupt as the KCS Communications interrupt.

Bits 7:6 are state bits that provide information that is used to ensure that the BMC and system software remain in sync with one another. Following are the possible states and their meaning:

Table 9-2, KCS Interface State Bits

S1 (bit 7)	S0 (bit 6)	Definition
0	0	IDLE_STATE. Interface is idle. System software should not be expecting nor sending any data.
0	1	READ_STATE. BMC is transferring a packet to system software. System software should be in the "Read Message" state.
1	0	WRITE_STATE. BMC is receiving a packet from system software. System software should be writing a command to the BMC.
1	1	ERROR_STATE. BMC has detected a protocol violation at the interface level, or the transfer has been aborted. System software can either use the "Get_Status' control code to request the nature of the error, or it can just retry the command.

Note: Whenever the BMC is reset (from power-on or a hard reset), the State Bits are initialized to "11 - Error State". Doing so allows SMS to detect that the BMC has been reset and that any message in process has been terminated by the BMC.

9.7.1 SMS_ATN Flag Usage

The SMS_ATN flag is used to indicate that the BMC requires attention from system software. This could either be because a message was received into the Receive Message Queue and ready for delivery to system software, the Event Message Buffer is full (if the Event Message Buffer is configured to generate an interrupt to system management software), a watchdog pre-timeout occurred, or because of an OEM event. Flags in the BMC identify which conditions are causing the SMS_ATN flag to be set. All conditions must be cleared (i.e. all messages must be flushed) in order for the SMS_ATN bit to be cleared.

The SMS_ATN bit is also used when the KCS interface is interrupt driven, or when OEM events or watchdog pre-timeouts generate a system interrupt. Refer to sections *9.12, KCS Communication and Non-communication Interrupts*, *9.13, Physical Interrupt Line Sharing*, and *9.14, Additional Specifications for the KCS interface* for additional information on the use and requirements for the SMS_ATN bit.

9.8 Command Register

The Command register must only be written from the system side when the IBF flag is clear. Only WRITE_START, WRITE_END, or GET_STATUS/ABORT Control Codes are written to the command register.

9.9 Data Registers

Packets to and from the BMC are passed through the data registers. These bytes contain all the fields of a message, such as the Network Function code, Command Byte, and any additional data required for the Request or Response message.

The Data_In register must only be written from the system side when the IBF flag is clear. The OBF flag must be set (1) before the Data_Out register can be read (see status register).

9.10 KCS Control Codes

The following table details the usage of 'Control Codes' by the KCS interface.

Table 9-3, KCS Interface Control Codes

Code	Name	Description	Target register	Output Data Register
60h	GET_STATUS / ABORT	Request Interface Status / Abort Current operation	Command	Status Code
61h	WRITE_START	Write the First byte of an Write Transfer	Command	N/A.
62h	WRITE_END	Write the Last byte of an Write Transfer	Command	N/A
63h-67h	reserved	reserved		
68h	READ	Request the next data byte	Data_In	Next byte
69h-6Fh	reserved	reserved		

Table 9-4, KCS Interface Status Codes

Code	Description
00h	No Error
01h	Aborted By Command (Transfer in progress was aborted by SMS issuing the Abort/Status control code)
02h	Illegal Control Code
06h	Length Error (e.g.overrun)
C0h-FEh	OEM Error (Error must not fit into one of above categories.)
FFH	Unspecified Error
all other	Reserved

9.11 Performing KCS Interface Message Transfers

System Management Software that uses the KCS Interface will typically be running under a multi-tasking operating system. This means transfers with the BMC may be interrupted by higher priority tasks or delayed by

other System Management Software processing. The SMS channel handshake is optimized to allow the BMC to continue to perform tasks between data byte transfers with System Management Software. The BMC does not time out data byte transfers on the SMS interface.

Request and Response Messages are paired together as a Write Transfer to the BMC to send the request followed by a Read Transfer from the BMC to get the response.

The process, as seen from the system perspective is depicted in *Figure 9-6, KCS Interface SMS to BMC Write Transfer Flow Chart,* and *Figure 9-7, KCS Interface BMC to SMS Read Transfer Flow Chart,* below.

During the write transfer each write of a Control Code to the command register and each write of a data byte to Data_In will cause IBF to become set, triggering the BMC to read in the corresponding Control Code or data byte.

If the KCS interface uses an interrupt, the BMC will write a dummy value of 00h to the output data register after it has updated the status register and read the input buffer. This generates an 'OBF' interrupt. The points at which this would occur are shown as "OBF" in *Figure 9-6, KCS Interface SMS to BMC Write Transfer Flow Chart*, below.

During the read phase, each write of a READ Control Code to Data_In will cause IBF to become set, causing the BMC to read in the Control Code and write a data byte to Data_Out in response. If the KCS interface uses an interrupt, the write of the data byte to Data_Out will also generate an interrupt. The point at which this would occur during the READ_STATE is shown as "OBF" in *Figure 9-7, KCS Interface BMC to SMS Read Transfer Flow Chart*, below.

Note that software does not need to use the Get Status/Abort transaction to return the interface to the IDLE_STATE or handle an error condition. The interface should return to IDLE_STATE on successful completion of any full command/response transaction with the BMC. Thus, since the interface will allow a command transfer to be started or restarted at any time when the input buffer is empty, software could elect to simply retry the command upon detecting an error condition, or issue a 'known good' command in order to clear ERROR_STATE.

9.12 KCS Communication and Non-communication Interrupts

The following lists some general requirements and clarifications to support both KCS communication and KCS non-communication interrupts on the same interrupt line using the OBF signal. A KCS communications interrupt is defined as an OBF-generated interrupt that occurs during the process of sending a request message to the BMC and receiving the corresponding response. This occurs from the start of the write (request) phase of the message (issuing WRITE_START to the command register) through to the normal conclusion of the corresponding read (response) phase of the message. (The conclusion of the communications interval is normally identified by the interface going to IDLE_STATE). KCS communications interrupts are also encountered during the course of processing a GET_STATUS/ABORT control code.

A KCS non-communication interrupt is defined as an OBF-generated interrupt that occurs when the BMC is not in the process of transferring message data or getting error status. This will typically be an interrupt that occurs while the interface is in the IDLE_STATE.

There are several options in the BMC that can be enabled to cause KCS non-communication interrupts as described in the *Set BMC Global Enables* command, and *Get Message Flags* commands. These are the watchdog timer pre-timeout interrupt, event message buffer interrupt, receive message queue interrupt, and the OEM interrupts. Software can detect which of the standard interrupts are supported by attempting to enable them using the *Set BMC Global Enable*s command and checking for an error completion code.

9.13 Physical Interrupt Line Sharing

A typical interrupt-driven implementation will assert a physical interrupt line when OBF is asserted. In order to allow a single interrupt line to serve for both communication and non-communication interrupts, the physical

interrupt line must be automatically deasserted by the BMC whenever a communication phase begins, even if there is a pending non-communications interrupt to be serviced. This is necessary so the interrupt line can be used for signaling communication interrupts . Once the communication operations have completed (return to idle phase) the controller must re-assert the interrupt line if the non-communications interrupt is still pending.

9.14 Additional Specifications for the KCS interface

This section lists additional specifications for the KCS interface.

- The BMC must generate an OBF whenever it changes the status to ERROR_STATE. This will ensure that any transition to ERROR_STATE will cause the interrupt handler to run and catch the state.
- The BMC generates an OBF upon changing the status to IDLE_STATE. An IPMI 1.5 implementation is allowed to share this interrupt with a pending KCS non-communication interrupt, or it elect to always generate a separate OBF interrupt for non-communications interrupts.
- A BMC implementation that elects to always generate a separate non-communications interrupt must wait for the OBF interrupt that signals entering the IDLE_STATE to be cleared before it asserts an OBF interrupt for the non-communications interrupt.
- IPMI v1.5 systems are allowed to generate a single OBF that covers both the last communications interrupt (when the BMC status goes to IDLE_STATE) and a pending non-communications interrupt. I.e. it is not required to generate a separate OBF interrupt for the non-communications interrupt if a non-communications interrupt was pending at the time the BMC status goes to IDLE_STATE. In order to support this, an *IPMI v1.5 KCS interface implementation must set SMS_ATN for all standard (IPMI defined) non-communication interrupt sources*.
- For IPMI v1.5, the BMC must set the SMS_ATN flag if any of the standard message flags become set. This includes Receive Message Available, Event Message Buffer Full (if the Event Message Buffer Full condition is intended to be handled by System Management Software), and Watchdog Timer pre-timeout flags, as listed in the *Get Message Flags* command. This is independent of whether the corresponding interrupt is enabled or not.
- The BMC must change the status to ERROR_STATE on any condition where it aborts a command transfer in progress. For example, if the BMC had an OEM command that allowed the KCS interface to be asynchronously reset via IPMB, the KCS interface status should be put into the ERROR_STATE and OBF set, not IDLE_STATE, in order for software to be notified of the change. However, the BMC does not change the status to the ERROR_STATE, but to the IDLE_STATE, when the BMC executes the Get Status/Abort control code from SMS I/F, even if the Get Status/Abort control code is used to abort a transfer.
- A cross-platform driver must be able to function without handling any of the OEM bits. Therefore, enabling SMS_ATN on OEM interrupts/states must not be enabled by default, but must be explicitly enabled either by the *Set BMC Global Enables* command or by an OEM-defined command.
- The SMS_ATN bit will remain set until all standard interrupt sources in the BMC have been cleared by the *Clear Message Flags* command, or by a corresponding command. For example, the Read Message command can automatically clear the Receive Message Queue interrupt if the command empties the queue.
- A KCS interface implementation that allows its interrupt to be shared with other hardware must set SMS_ATN whenever it generates a KCS interrupt. A system will typically report whether it allows an interrupt to be shared or not via resource usage configuration reporting structures such as those in ACPI.
- OEM non-communications interrupts should be disabled by default. They must be returned to the disabled state whenever the controller or the system is powered up or reset. This is necessary to allow a generic driver to be used with the controller. A driver or system software must be explicitly required to enable vendor-specific non-communications interrupt sources in order for them to be used. OEM non-communications interrupt sources must not contribute to SMS_ATN when they are disabled.
- The OEM 0, 1, and 2 flags that are returned by the *Get Message Flags* command may also cause the SMS_ATN flag to be set. A platform or system software must not enable these interrupts/flags unless there is a

corresponding driver that can handle them. Otherwise, a generic cross-platform driver could get into a situation where it would never be able to clear SMS_ATN.

- It is recommended that any OEM generated non-communications interrupts cause at least one of the OEM flags in the *Get Message Flags* to become set. This will enable improving system efficiency by allowing a cross-platform driver to pass the value of the *Get Message Flags* to an OEM extension, saving the OEM extension software from having to issue an additional command to determine whether it has an anything to process.
- It is recommended that an OEM that uses the OEM flags sets the SMS_ATN flag if one or more of the OEM flags (OEM 0, OEM 1, or OEM 2) becomes set, especially if those flags can be the source of a KCS non-communications interrupt. The driver can use SMS_ATN as the clue to execute the *Get Message Flags* command and pass the data along to an OEM extension routine.
- OEM non-communications interrupts may elect to either share the IDLE_STATE OBF interrupt with the non-communications interrupt OBF, or generate a separate non-communications OBF interrupt. If the OEM non-communications interrupt implementation shares the IDLE_STATE OBF interrupt, the OEM non-communications interrupt must also set SMS_ATN.

9.15 KCS Flow Diagrams

The following flow diagrams have been updated from corresponding diagrams in the original IPMI v1.0, rev. 1.1 specification. This information applies to the following flow diagrams:

- All system software wait loops should include error timeouts. For simplicity, such timeouts are not shown explicitly in the flow diagrams. A five-second timeout or greater is recommended.
- The phase values represent state information that could be kept across different activations of an interrupt handler, and corresponding entry points. Based on the 'phase' the interrupt handler would branch to the corresponding point when an OBF interrupt occurred. The information may also be useful for error reporting and handling for both polled- and interrupt-driven drivers. Note that other state may need to be kept as well. For example, during the 'wr_data' phase, the handler may also need to preserve a byte counter in order to track when the last byte of the write was to be sent.
- The symbol of a circle with an arrow and the text 'OBF' inside the circle represents the points where the BMC would write a dummy data byte to the output buffer in order to create an OBF interrupt. The label above the circle indicates where an interrupt handler would branch to when the OBF interrupt occurs under in the corresponding phase. An interrupt handler would exit upon completing the step that occurs before where the OBF interrupt symbol points.

Figure 9-6, KCS Interface SMS to BMC Write Transfer Flow Chart

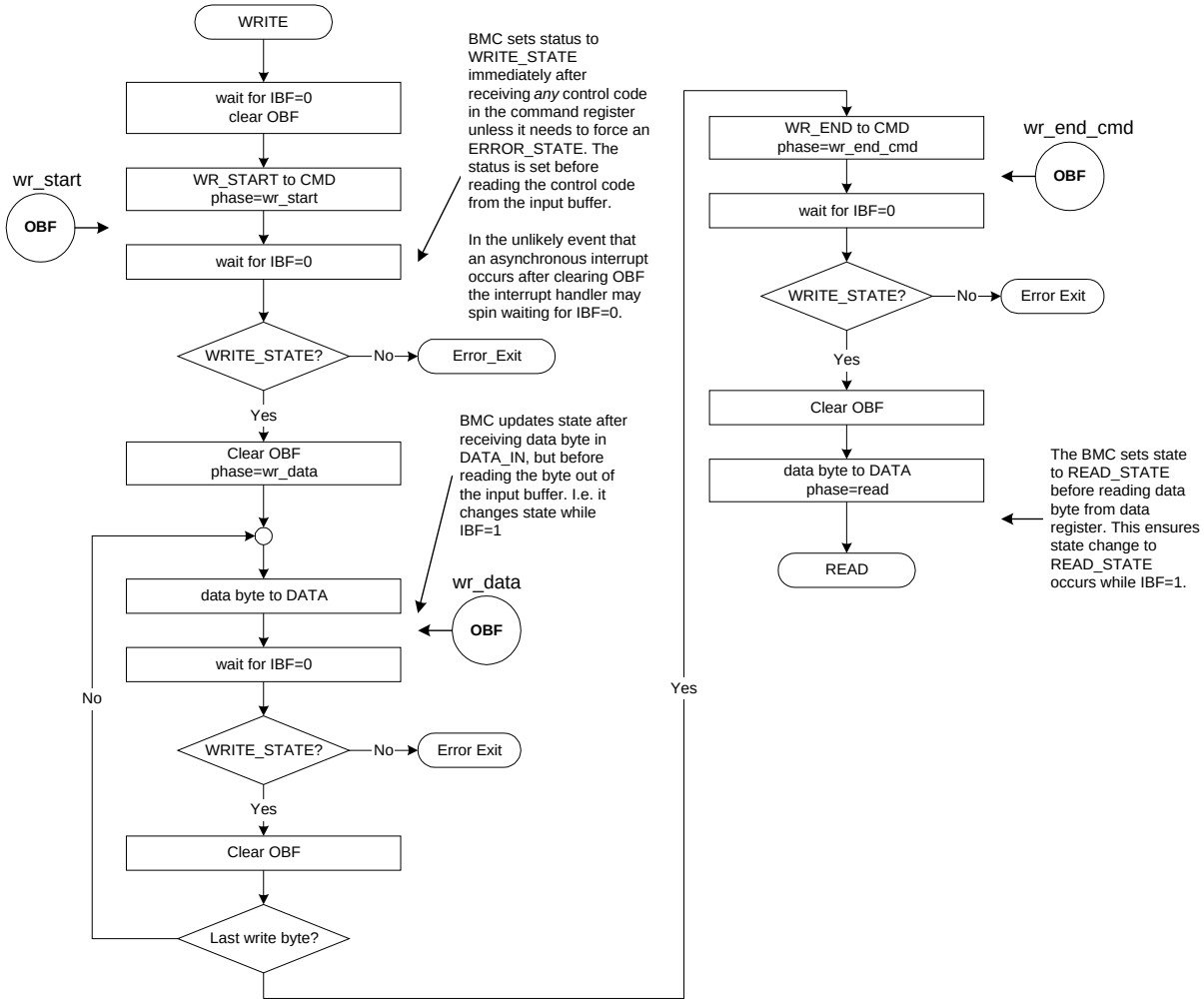
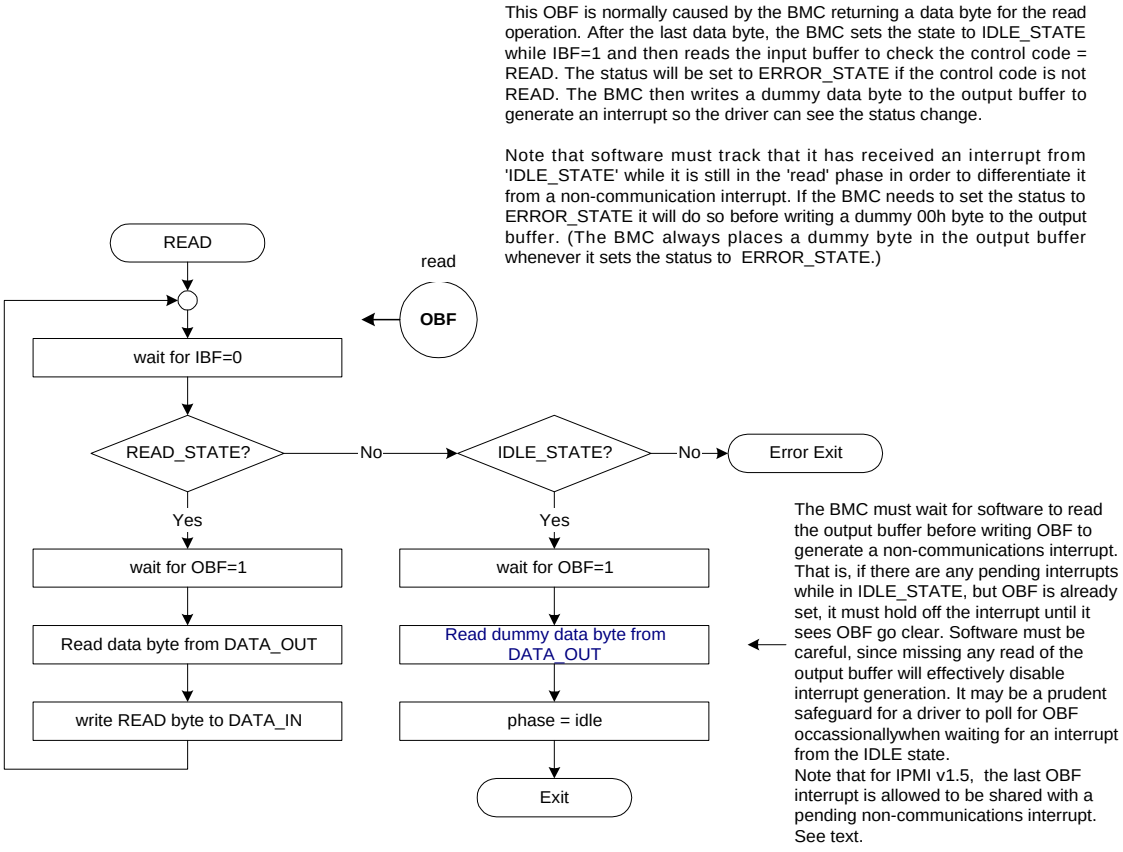
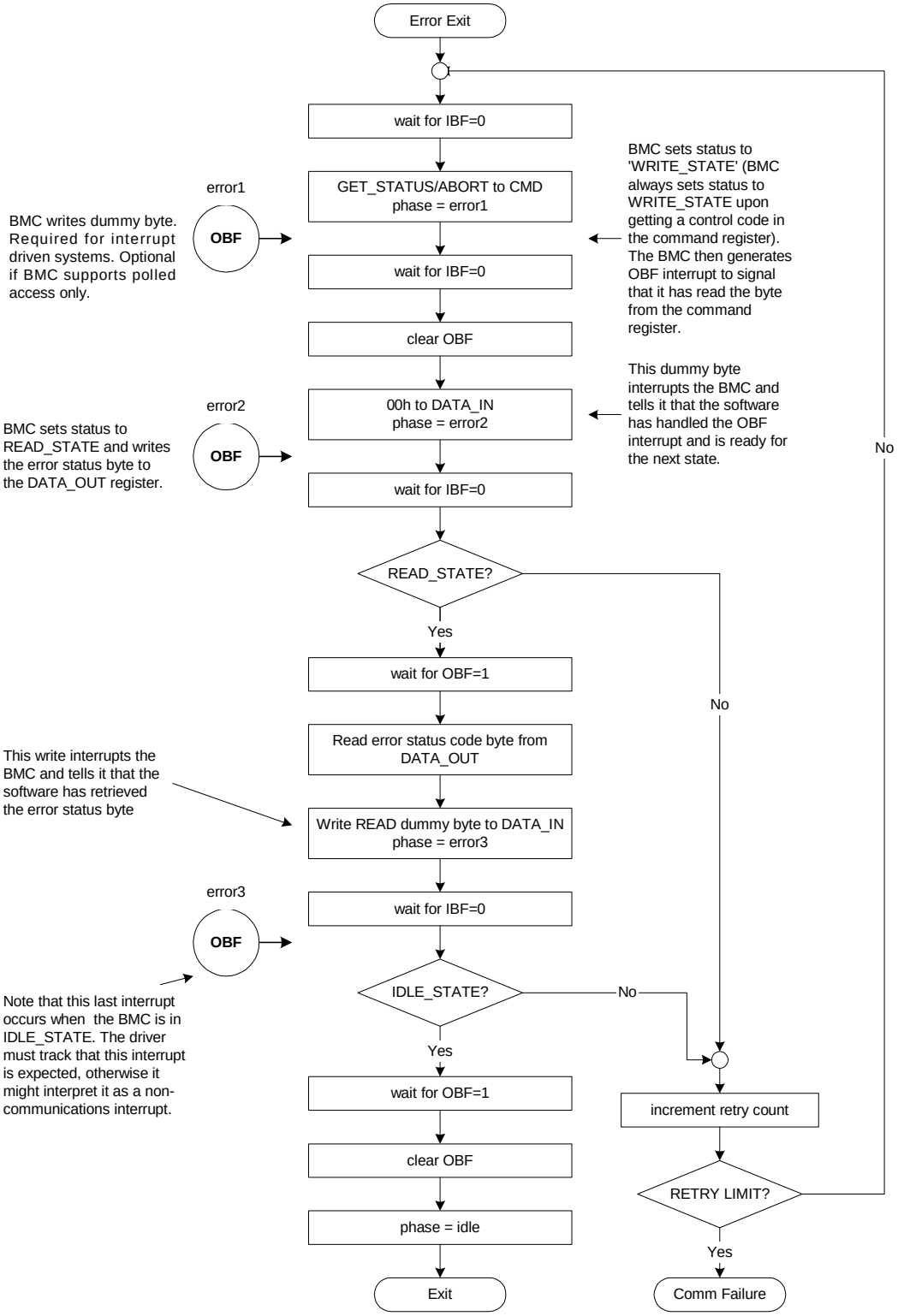


Figure 9-7, KCS Interface BMC to SMS Read Transfer Flow Chart



The following figure shows a flow diagram for aborting KCS transactions in progress and/or retrieving KCS error status.

Figure 9-8, Aborting KCS Transactions in-progress and/or Retrieving KCS Error Status



9.16 Write Processing Summary

The following summarizes the main steps write transfer from system software to the BMC:

- Issue a ‘WRITE_START’ control code to the Command register to start the transaction.
- Write data bytes (NetFn, Command, Data) to Data_In.
- Issue an ‘WRITE_END’ control code then the last data byte to conclude the write transaction.

9.17 Read Processing Summary

The following summarizes the main steps for a read transfer from the BMC to system software:

- Read Data_Out when OBF set
- Issue READ command to request additional bytes
- If READ_STATE (after IBF = 0), repeat previous two steps.

9.18 Error Processing Summary

The following summarizes the main steps by which system software processes KCS Interface errors:

- Issue a ‘GET_STATUS/ABORT’ control code to the Command register. Wait for IBF=0. State should be WRITE_STATE.
- If OBF=1, Clear OBF by reading Data_Out register.
- Write 00h to data register, wait for IBF=0. State should now be READ_STATE.
- Wait for OBF=1. Read status from Data_Out
- Conclude by writing READ to data register, wait for IBF=0. State should be IDLE.

9.19 Interrupting Messages in Progress

If, during a message transfer, the system software wants to abort a message it can do so by the following methods:

1. Place another "WRITE_START" command into the Command Register (a WRITE_START Control Code is always legal). The BMC then sets the state flags to "WRITE_STATE" and sets its internal flags to indicate that the stream has been aborted.

2. Send a "GET_STATUS/ABORT" request. This is actually the same as #1 above but is explicitly stated to indicate that this command will cause the current packet to be aborted. This command allows a stream to be terminated and the state to be returned to IDLE without requiring a complete BMC request and response transfer.

9.20 KCS Driver Design Recommendations

- A generic, cross-platform driver that supports the interrupt-driven KCS interface is not required to handle interrupts other than the interrupt signal used for IPMI message communication with the BMC. The message interrupt may be shared with other BMC interrupt sources, such as the watchdog timer pre-timeout interrupt, the event message buffer full interrupt, and OEM interrupts.

- A cross-platform driver should use the *Get BMC Global Enables* and *Set BMC Global Enables* commands in a 'read-modify-write' manner to avoid modifying the settings of any OEM interrupts or flags.

- It is recommended that cross-platform driver software provide a 'hook' that allows OEM extension software to do additional processing of KCS non-communication interrupts. It is highly recommended that the driver execute the *Get Message Flags* command whenever SMS_ATN remains set after normal processing and provide the results to the OEM extension software.

- The driver cannot know the whether the pre-existing state of any OEM interrupts or flags is correct. Therefore, a driver that supports OEM extensions should allow for an OEM initialization routine that can configure the OEM flags/interrupts before KCS OBF-generated interrupts are enabled.

- It is recommended that cross-platform drivers or software make provision for BMC implementations that may miss generating interrupts on a command error condition by having a timeout that will activate the driver or software in case an expected interrupt is not received.

- A driver should be designed to allow for the possibility that an earlier BMC implementation does not set the SMS_ATN flag except when there is data in the Receive Message Queue. If the driver cannot determine whether SMS_ATN is supported for all enabled standard flags or not, it should issue a *Get Message Flags* command whenever it gets a KCS non-communication interrupt.

- A driver or system software can test for whether the Watchdog Timer pre-timeout and/or Event Message Buffer Full flags will cause SMS_ATN to become set. This is accomplished by disabling the associated interrupts (if enabled) and then causing a corresponding action that sets the flag. This is straightforward by using the watchdog timer commands in conjunction with the *Set BMC Global Enables* and *Get Message Flags* commands.

  For example, to test for the Event Message Buffer Full flag setting SMS_ATN, first check to see if the Event Message Buffer feature is implemented by attempting to enable the event message buffer using the *Set and Get BMC Global Enables* command. If the feature is not implemented, an error completion code will be returned. Next, disable event logging and use the watchdog timer to generate an SMS/OS 'no action' timeout event, then see if the SMS_ATN becomes set. If so, use the *Get Message Flags* command to verify that the Event Message Buffer Full flag is the only one set (in case an asynchronous message came in to the Receive Message Queue during the test.) The pre-timeout interrupt can be testing in a similar manner.

- It is possible (though not recommended) for a BMC implementation to include proprietary non-communication interrupt sources that do not set SMS_ATN. These sources must not be enabled by default. It is recommended that a generic cross-platform driver have provisions for OEM extensions that get called whenever a non-communication interrupt occurs. It is recommended that the extension interface provides the last reading of the KCS flags so that an OEM extension can see the state of SMS_ATN.
- Software should be aware that IPMI v1.0 implementations were not required to set SMS_ATN for all non-communication interrupts. If a BMC implementation does not set SMS_ATN for all non-communication interrupts, it must generate a separate OBF interrupt for non-communication interrupts. A controller that does not set SMS_ATN for all non-communication interrupts is not allowed to use the same OBF interrupt to signal the both completion of communications and a non-communications interrupt.
- Regardless of whether the IDLE_STATE OBF interrupt is shared with a pending non-communications interrupt, software drivers must examine SMS_ATN after clearing OBF. If SMS_ATN is asserted the driver must process the non-communications interrupt sources.

10. SMIC Interface

This section provides the specifications of the SMIC (Server Management Interface Chip) interface. The SMIC interface is one of the physical interfaces specified for transferring IPMI messages between the system management software and the system's primary management controller (BMC).

The interface can be readily implemented using an external ASIC or standard programmable logic to provide a byte I/O-mapped messaging interface to standard microcontrollers.

The SMIC Interface is designed to support polled operation. Implementations can optionally provide an interrupt driven from the BUSY bit, but this must not prevent driver software from using the interface in a polled manner. This allows software to default to polled operation. It also allows software to use the KCS interface in a polled mode until it determines the type of interrupt support. Methods for assigning and enabling such an interrupt are outside the scope of this specification.

The specification of the SMIC interface registers is given solely with respect to the 'system software side' view of the interface in system I/O space.

The functional behavior of the management controller to support the SMIC registers is specified, but the physical implementation of the interface and the organization of the interface from the management controller side is implementation dependent and is beyond the scope of this specification.

10.1 SMS Transfer Streams

The SMIC interface is designed to be interruptible to allow the one physical interface to be shared by two types of system software: SMM (System Management Mode) software that runs from within an SMI Handler, and SMS (System Management Software) that runs under the OS.

If an SMS transaction is interrupted, system management software will need to restart the Request/Response transaction it had in progress.

To support this sharing, the interface provides mechanisms that allow system management software to detect that its use of the interface has been interrupted. The protocol for messaging between SMM and the BMC over the SMIC interface is implementation specific and not covered by this specification.

10.2 SMIC Communication Register Overview

The SMIC registers are mapped into system I/O space. This shared register space consists of three byte-wide registers:

- Flags Register - provides flags for use in various defined operations
- Control/Status Register - accepts control codes and returns status codes
- Data Register - provides a port for transactions that exchange message data

Message contents are passed through the data register. This includes the fields of a message, such as the Network Function code, Command Byte, and any additional data required for the Request or Response message.

The control register is loaded with *control code* values that are used for framing the message data (indicating message start, middle, and end) and for indicating message data transfer direction.

Status codes are returned through the control/status register. A control code is required to initiate for each data byte transferred through the data register.

The Flags register contains bits that indicate whether the controller has a message for system software, generated an SMI, or is ready for a transfer operation. The Flags register also contains a special BUSY bit, that is used by system software to initiate and handshake data byte transfers through the interface.

The SMIC interface is used as a polled interface. System software is always the “Master” for transfers between system software and the BMC. The BMC can signal that data is available via bits in the flags register, but data bytes will not be moved to or from the data register until the transaction is initiated by system software.

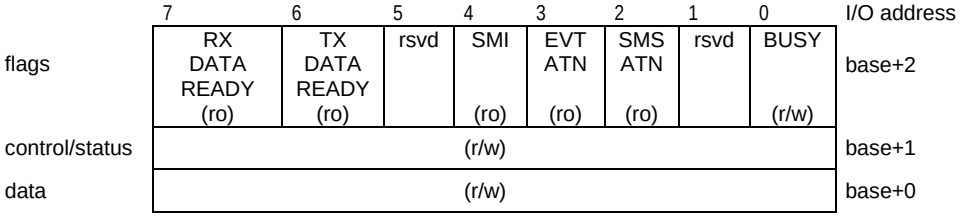
10.3 SMIC/BMC Message Interface Registers

The following figure illustrates the SMIC/BMC Interface Registers and register bits. These registers are located at three consecutive 8-bit port addresses in I/O space.

The data, control/status, and flags registers appear at an I/O addresses **0CA9h, 0CAAh, and 0CABh**, respectively.

Reserved bits should be written as ‘0’ and ignored during reads. Software should not assume that a reserved bit will return a constant value.

Figure 10-1, SMIC/BMC Interface Registers



10.3.1 Flags Register

System software always initiates the SMIC transfers, regardless of direction. The management controller uses the SMS_ATN bit in the Flags register to indicate to system software that is has a message to be read. This bit will be set whenever data is present in the Receive Message Queue.

Other bits in the Flags register are used to arbitrate access to the control/status and data registers between the BMC and system software. Bits 7::2 are read-only from the system bus and write-only from the BMC; these bits are used by the BMC as communication and status flags. Bit 0, the BUSY bit, is used as a semaphore for coordinating access to the control/status and data registers between the system bus and the BMC. The following table summarizes the functions of Flags Register bits.

Table 10-1, SMIC Flags Register Bits

Bit	Name	Description
7	RX_DATA_RDY	Indicates that the BMC has data that can be delivered in response to a 'READ' control code. RX_DATA_RDY must be '1' before a control code that causes a data byte to be transferred (read) from the BMC into the data register can be issued. RX_DATA_RDY shall also be set to '1' whenever a 'READY' status code is returned. *RX_DATA_RDY should be ignored when issuing control codes that do not cause a data byte to be read from the BMC.*
6	TX_DATA_RDY	Indicates that the BMC is ready to accept a 'WRITE' control code and data. TX_DATA_RDY must be '1' before a control code that causes data to be transferred (written) to the BMC from the data register can be issued. *TX_DATA_RDY can be ignored when issuing control codes that do not cause a data byte to be written to the BMC.*
5	Reserved	
4	SMI	Indicates that the BMC has asserted the SMI signal and has internal 'SMI event' flags that are set.
3	EVT_ATN	Indicates that an Event Message has been received by the BMC and is ready to be read from the Event Message Buffer in the BMC. If SMIs are used, this flag may also set when an OEM message for the SMI Handler is available.
2	SMS_ATN	Indicates that the BMC has messages that are ready to be read from the Receive Message Queue in the BMC. An implementation can use the 0-to-1 transition of this bit to provide an interrupt. Clearing the Receive Message Queue clears this bit and clears 'Receive Message Queue not empty' as an interrupt source.
1	Reserved	
0	BUSY	Provides the arbitration mechanism for SMIC mailbox register access. This bit is only set (1) from the system side and only cleared (0) by the BMC. The system side sets the BUSY bit whenever it wishes to send a control code (and data, if appropriate) to the BMC. The BMC acknowledges that it has accepted and acted on the control code (and performed the data byte transfer) by clearing the BUSY bit. An implementation can use the 1-to-0 transition of BUSY to provide an interrupt.

10.3.2 Control/Status Register

The Control/Status register is the destination for control codes written from the system bus, and Status codes returned by the BMC.

SMS transfers have a specific numeric range for control codes and status codes. This provides a 'stream ID' that allows the BMC to tell whether SMS or some other message stream issued a control code. This also allows system software to detect interruption by examining which the range of values for the most recent status code. The control and status codes used for SMS transactions are defined in the control code and status code tables in the sections following *Section 10.9, SMIC Control and Status Code Ranges.*

10.3.2.1 Control and Status Codes

Message transfer control and framing codes (control codes) are transferred via the Control/Status register while message content, such as command and data bytes, is transferred the Data register. control codes are unique to each transfer stream and defined transaction and for each phase (beginning, intermediate, and end) of a message.

Status Codes confirm the message phasing, identify the active stream, and provide error status.

When a message is transferred between system software and the BMC, each byte of the message that is passed through the data register is accompanied by a control code written to the Control/Status register. The BMC acknowledges reception of the control code and data byte by writing a corresponding Status Code to the

Control/Status register before clearing the BUSY bit. Note that Status Codes are returned for each transaction, regardless of whether a data byte is transferred or not.

10.3.3 Data Register

The message bytes for all requests (commands) and responses between system software and the BMC pass through the Data register. The data register must only be written or read from the system side when the BUSY bit is clear.

Messages to the BMC contain the same types of message body fields as messages on the IPMB. This includes Network Function, Command byte, and Data fields.

10.4 Performing a single SMIC/BMC Transaction

The following steps describe how system software issues a control code to the BMC and transfers a data byte through the SMIC interface.

1. System software polls the BUSY bit of the Flags register until it reads back as cleared (0) by the BMC. When the BUSY bit is 0, the BMC is ready to accept a new control code. System software is not allowed to access the Control/status or Data Registers when the BUSY bit is high.
2. System software writes the control code to the Control/Status register. See *Section 10.9, SMIC Control and Status Code Ranges* and following, for control and status code specifications.

   If the transfer is a ‘Write’ transfer, and the control code is for ‘WR_NEXT’ or ‘WR_END’ operation, system software waits for the TX_DATA_RDY bit to be set become set. This indicates that the BMC is ready for the next write data byte. If the transfer is a ‘Read’ transfer, wait for the RX_DATA_RDY bit to be set. (The exception to this is the ‘GET_STATUS’ control code, which though it causes a data byte to be returned (the error code) does not require RX_DATA_RDY to be high first.
3. If the transfer is a ‘Write’ transfer, system software loads the data to be written to the BMC into the Data register.
4. System software then initiates the operation by setting the BUSY bit. Setting the BUSY bit causes the BMC to read the SMIC control code register and act on the control code. If the transfer is a Write transfer, the BMC reads the data from the data register at this time. If the transfer is a Read transfer, the BMC writes the data to the data register. The controller then returns a status code in the control/status register. data or an error code in the data register (as appropriate), and clears the BUSY bit.
5. System software waits for the BUSY bit to clear, indicating the completion of the control code operation.
6. System software reads the Control/Status register for the completion status of the transaction. If the operation was successful, the status code will reflect the next step in the transaction, or the successful completion of the transaction. If the operation was not successful, the status code will be set to ‘READY’ and the data register will hold an error code.

10.5 Performing a SMIC/BMC Message Transfer

Multiple transactions are required to transfer a message between system software and the BMC. In this case, a message transfer refers to the sequence of steps required to transfer a series of data bytes to or from the BMC. One control code transaction is required for each message data byte transferred via the SMIC interface.

A message transfer can be restarted at any time. Issuing an SMS_WR_START control code immediately aborts any message transfer in progress and begins a new write transfer. Issuing WRITE_START control codes does not require the RX_DATA_RDY or TX_DATA_RDY flags to be set.

The control code/status code sequences for the SMS-to-BMC transactions follows a "*Transfer Start, Transfer Middle, Transfer End*" pattern:

- Issue a 'Start' control code to start the transaction. Signifying *'Transfer Start'*
- Issue 'Next' control codes to transfer the body of the data bytes. Signifying *'Transfer Middle'*. These are either 'Write_Next' or 'Read_Next' control codes, dependent on the transfer direction.
- Issue an 'End' control code to conclude the transaction and return the stream to the 'Ready' status. This signifies *'Transfer End'*

The following summarizes these steps:

1. If the transfer is a write transfer (system software to BMC), load the data register with the appropriate data and issue the 'Write Start' control code for the stream. There is no need to check TX_DATA_RDY.

   If the transfer is a read transfer (a data byte transfer from the BMC to system software) wait for the RX_DATA_RDY flag to become set, then issue the 'Read Start' control code for the stream.

2. After each transaction, check the status code to see if the operation was successful. For write transfers, the status code will generally be a 'Write Next', indicating that the interface is ready to accept more data. For read transfers, the status code will typically be either a 'Read Next', indicating that there is more data to be read, or a 'Read End' indicating that the last byte of data was transferred. If the operation was aborted or an error occurred the transaction will need to be restarted from the beginning.

3. Continue the transfer based on the status code. For read transfers, wait for the RX_DATA_RDY flag and perform read operations until a 'Read End' status code is encountered (or an abort). For write transfers, wait for the TX_DATA_RDY flag and perform write operations until you conclude the transfer with a 'Write End' control code.

4. Issue any additional control codes to return the transfer stream to the 'Ready' condition (indicated by the 'Ready' status code). For read transfers from the SMM/SMS streams, this requires issuing a 'Read End'.

10.6 Interrupting Streams in Progress

Any software that interrupts a transfer in progress and switches to another stream is responsible saving and restoring the status and data register values for the transaction that was in effect at the time of the interrupt. The interrupting routine must first wait for the BUSY bit to clear and then save the control/status and data register contents. Before exiting, the interrupting routine must wait for the BUSY bit to clear following its last transaction, then restore the control/status and data register values. The interrupting routine can then perform its transfer(s). After the interrupting routine concludes its last transaction, it must wait for BUSY to clear and restore the original control/status and data register contents before returning from the interrupt.

The following summarizes the steps for an interrupting routine, e.g. an SMI Handler:

1. Poll the BUSY bit until cleared by the BMC.
2. Save the contents of the Control/Status and Data registers.
3. Perform the desired message transfers.
4. Wait for the BUSY bit to clear, then restore the Control/Status and Data register values that were saved in step 2, and return to the interrupted routine. If the interrupted routine has additional bytes to transfer, the succeeding control code will be 'out-of-phase' with the state expected by the BMC. The BMC will then return a 'READY' status code with an 'Aborted' return value. This indicates to the interrupted routine that it needs to restart the transaction. If the interrupt happened to occur between transfers, or on the last transfer of a transaction, the Control/Status and Data registers will have the correct values and the interrupted routine will be able to start a new transaction.

10.7 Stream Switching

A stream switch occurs when the BMC receives a WR_START control code with a 'stream ID' that is different than the stream ID for the previous control code. This is the mechanism that SMS uses to restart an interrupted transaction. If a control code, other than WR_START is issued, and the stream does not match the stream ID for the previous control code, the BMC shall return a 'READY' status code with an 'Aborted' return value.

10.8 DATA_RDY Flag Handling

The BMC shall set the TX_DATA_RDY whenever it is ready to accept a 'WR_START', 'WR_NEXT', or 'WR_END' control code that transfers a data byte from the SMIC data register. Note that system software does not need to check for TX_DATA_RDY in order to issue a WR_START.

The BMC shall set the RX_DATA_RDY flag whenever it has a data byte that is ready to be requested with a 'Read Start', 'Read Next', or 'Read End' control code, or when it is prepared to return a 'Ready' status code.

> *The BMC shall set the RX_DATA_RDY flag whenever it returns a 'Ready' status code to the stream. This includes when a stream is interrupted or when other errors occur during a transfer. This is to ensure that a routine that may be spinning on the RX_DATA_RDY bit will proceed and attempt its next transaction.*

The BMC shall only *deassert (0)* the RX_DATA_RDY or TX_DATA_RDY flags *while BUSY is asserted (1).* The BMC can *assert* the RX_DATA_RDY or TX_DATA_RDY flags *any time* that the associated conditions become true.

10.9 SMIC Control and Status Code Ranges

Specific Control Code ranges are used to identify transactions using the SMS transfer stream. This allows the BMC to tell when the SMS stream is in use. Status Codes are returned by the BMC in the SMIC Control/Status register to reflect the completion status of a previously issued control code. Like the control codes, the Status Codes occupy a specific range for SMS transactions. Another set of control and status code ranges is reserved for OEM / SMI Handler.

The presently defined ranges are:

- **40h-5Fh** SMS (System Management Software) Transfer Stream Control Codes
- **C0h-DFh** SMS (System Management Software) Transfer Stream Status Codes
- **60h-7Fh** Available SMM (System Management Mode) / OEM Transfer Stream Control Codes
- **E0h-FFh** Available SMM (System Management Mode) / OEM Transfer Stream Status Codes

All unspecified codes are *reserved*.

10.10 SMIC SMS Stream Control Codes

Table 10-2, SMS Transfer Stream control codes

Code	Name	Description
		SMS STREAM CONTROL CODES
40h	CC_SMS_GET_STATUS	Get status related to the SMS (system mgt. software) transfer stream. An 'SC_SMS_RDY' status code will be returned as the completion status for this control code, along with the last error code for the stream in the data register.
41h	CC_SMS_WR_START	Write the first message byte of an SMS write transfer. This is also used to switch to the SMS stream. The SMIC data register must be loaded with the data byte to be written to the BMC. The non-error completion status for this control code will be an 'SC_SMS_WR_START' status code. The data register contents will remain unaltered if no error occurred.
42h	CC_SMS_WR_NEXT	Write a 'middle' message byte in an SMS write transfer. The SMIC data register must be loaded with the data byte to be written to the BMC. The user must wait for the TX_DATA_RDY=1 before issuing the control code. The non-error completion status for this control code will be an 'SC_SMS_WR_NEXT' status code. The data register contents will also remain unaltered if no error occurred.
43h	CC_SMS_WR_END	Indicates the last message byte for an SMS write transfer. The SMIC data register must be loaded with the last data byte to be written to the BMC for the current message. The user must wait for TX_DATA_RDY=1 before issuing this control code. The non-error completion status for this control code will be an 'SC_SMS_WR_END' status code with an error code of '00' in the data register, indicating 'OK'.
44h	CC_SMS_RD_START	Get the first byte of a read transfer from the BMC. The user must wait for RX_DATA_RDY = 1 before issuing the control code. The non-error completion status for this control code will be an 'SC_SMS_RD_START' status code in the status register and the data byte in the data register.
45h	CC_SMS_RD_NEXT	Get a 'middle' message byte for an SMS read transfer. The user must wait for RX_DATA_RDY = 1 before issuing the control code. The non-error completion status for this control code will be an 'SC_SMS_RD_NEXT' status code in the status register if there is more data to read or an 'SC_SMS_RD_END' status code if the last byte was transferred, and the data byte in the data register.
46h	CC_SMS_RD_END	Used to tell the BMC that the last byte of an SMS read transfer has been read from the data register. It is not necessary to check the RX_DATA_RDY flag before performing this operation. The non-error completion status for this control code will be an 'SC_SMS_RDY' status code with an error code of '00' in the data register, indicating 'OK'.
47h-5Fh	reserved	reserved

10.11 SMIC SMS Stream Status Codes

Table 10-3, SMS Transfer Stream Status Codes

Code	Name	Description
		SMS STREAM STATUS CODES
C0h	SC_SMS_RDY	BMC is ready for next SMS transfer. An error code is returned in the data register: 00 = NO ERROR 01 = UNSPECIFIED ERROR / ABORTED 02 = ILLEGAL or unexpected control code 03 = NO RESPONSE - response timeout. This will occur if the BMC cannot supply a command response. 04 = ILLEGAL command. The request message is not recognized as being a legal BMC request. 05 = BUFFER FULL. Attempt to write too many bytes to the BMC.
C1h	SC_SMS_WR_START	This status code indicates that the BMC has accepted first byte of a write transfer and is ready for the next transaction.
C2h	SC_SMS_WR_NEXT	The BMC has accepted next data byte of the write transfer, and is ready for the next transaction.
C3h	SC_SMS_WR_END	The BMC has accepted the byte as being the last byte of the write transfer and is ready for next SMS transfer. An error code is returned in the data register: 00 = NO ERROR 01 = ABORTED 02 = ILLEGAL or unexpected control code 03 = NO RESPONSE - response timeout. This will occur if the BMC cannot supply a command response. 04 = ILLEGAL command. The request message is not recognized as being a legal BMC request. 05 = BUFFER FULL. Last byte could not be accepted.
C4h	SC_SMS_RD_START	BMC has accepted the start of an SMS stream read transfer. The first data byte of the read transfer is returned in the data register.
C5h	SC_SMS_RD_NEXT	The BMC acknowledges a CC_SMS_RD_NEXT control code and is indicating that there is more data to be read. The requested data byte is in the data register.
C6h	SC_SMS_RD_END	The BMC acknowledges a CC_SMS_RD_NEXT control code and is indicating that there is no more data to be read. The last data byte is in the data register.
C7h-DFh	reserved	reserved

10.12 SMIC Messaging

The SMIC message interface is essentially a 'single master' interface, where the 'Master' is the system software on the system side of the interface. System software can only write Request Messages to the BMC, and can only receive Response Messages from the BMC.

This does not mean that the system software cannot receive downstream 'requests' from the IPMB, or even the BMC. Downstream requests can be 'wrapped' in a BMC Response Message. For example, a downstream request could be placed in the Receive Message Queue where the data is retrieved using the *Get Message* command. The Response Message would contain the downstream request data - which could then be extracted from the Response Message by system software.

Since the SMIC interface is a 'point-to-point' connection, a 'Requester's ID' is not required in a Request Message to identify which physical interface to return a message response to. Only SMIC Event Request messages include a Requester ID in the form of the Software ID field.

10.13 SMIC/BMC LUNs

LUN 00b is typically used for all messages to the BMC through the SMIC interface. LUNs 01b is reserved for Receive Message Queue use and should not be used for sending other commands to the BMC. Note that messages encapsulated in a *Send Message command* can use any LUN in the encapsulated portion.

10.14 SMIC-BMC Request Message Format

Request Messages are sent to the BMC from system software using a *write transfer* through the SMIC. The message bytes are organized according to the following format specification:

Figure 10-2, SMIC/BMC Request Message Format

Byte 1	Byte 2	Byte 3:N
NetFn/LUN	Cmd	Data

Where:

LUN Logical Unit Number. This is a sub-address that allows messages to be routed to different 'logical units' that reside behind the same physical interface. The LUN field occupies the least significant two bits of the first message byte.

NetFn Network Function code. This provides the first level of functional routing for messages received by the BMC via the SMIC interface. The NetFn field occupies the most significant six bits of the first message byte.

Cmd Command code. This message byte specifies the operation that is to be executed under the specified Network Function.

Data Zero or more bytes of data, as required by the given command. The general convention is to pass data LS-byte first, but check the individual command specifications to be sure.

10.15 BMC-SMIC Response Message Format

Response Messages are *read transfers* from the BMC to system software via the SMIC. Note that the BMC only returns responses via the SMIC interface when Data needs to be returned. The message bytes are organized according to the following format specification:

Figure 10-3, SMIC/BMC Response Message Format

Byte 1	Byte 2	Byte 3	Byte 4:N
NetFn/LUN	Cmd	Completion Code	Data

Where:

LUN Logical Unit Number. This is a return of the LUN that was passed in the Request Message.

NetFn Network Function. This is a return of the NetFn code that was passed in the Request Message.

Cmd Command. This is a return of the Cmd code that was passed in the Request Message.

Completion Code The Completion Code indicates whether the request completed successfully or not.

Data Zero or more bytes of data. The BMC always returns a response to acknowledge the request, regardless of whether data is returned or not.

10.16 Logging Events from System Software via SMIC

The SMIC interface can be used for sending Event Messages from system software to the BMC Event Receiver. The following figures show the format for SMIC Event Request and corresponding Event Response messages. Note that only Event Request Messages to the BMC via the SMIC interface have a Software ID field. This is so the Software ID can be saved in the logged event.

Figure 10-4, SMIC Event Request Message Format

NetFn (04h = Sensor/Event Request)	LUN (00b)	Command (02h = Platform Event)	Software ID (Gen ID), 7-bits (20h-2Fh = system sw)	1			
EvMRev	Sensor Type	Sensor #	Event Dir	Event Type	Event Data 1	Event Data 2	Event Data 3

Shading designates fields that are not stored in the event record.

Figure 10-5, SMIC Event Response Message Format

NetFn (05h = Sensor/Event Response)	00	Command (02h = Platform Event)	Completion Code

11. Block Transfer (BT) Interface

This section describes the Block Transfer (BT) Interface. The BT interface is one of the supported BMC to SMS system interfaces. The BT interface is specified for SMS or OEM Defined messages. Messaging between the BMC and an SMI Handler is not specified for this interface.

The BT Interface is so named because an entire block of message data is buffered before the management controller is notified of available data. This is different from the SMIC and KCS interfaces, which are byte-transfer oriented. A *BT Interface Capabilities* command provides supplementary information about extended buffer sizes and other elements of the interface.

The host side of the BT Interface is designed for interrupt or polled operation. Implementations can elect to provide a system interrupt from the assertion of the B2H_ATN or SMS_ATN (BMC-to-Host attention or System Management Software attention) states. Note that implementing an interrupt must not preclude driver software from the using the interface in a polled manner.

> *The BT Interface is designed for efficient interrupt operation via assertion of H2B_ATN by the host. Provision for operation in a polled mode is optional.*

Methods for assigning, enabling, and determining the system interrupt are outside the scope of this specification.

The BT interface provides support for implementations that allow the submission and asynchronous completion of commands.

11.1 BT Interface-BMC Request Message Format

Request Messages are sent to the BMC from system software using a *write transfer* through the BT Interface. The message bytes are organized according to the following format specification:

Figure 11-1, BT Interface/BMC Request Message Format

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5:N
Length	NetFn/LUN	Seq	Cmd	Data

Where:

Length This is not actually part of the message, but part of the *framing* for the BT Interface. This value is the 1-based count of message bytes following the length byte. The minimum length byte value for a command to the BMC would be 3 to cover the NetFn/LUN, Seq, and Cmd bytes.

LUN Logical Unit Number. This is a sub-address that allows messages to be routed to different 'logical units' that reside behind the same physical interface. The LUN field occupies the least significant two bits of the first message byte.

NetFn Network Function code. This provides the first level of functional routing for messages received by the BMC via the BT Interface. The NetFn field occupies the most significant six bits of the first message byte.

Seq Used for matching responses up with requests. The BT interface can support interleaved 'multi-threaded' communications. There can be multiple simultaneous outstanding requests from SMS with responses returned asynchronously (and in any order). The Requester (SMS) sets the value for this field. The Responder returns the value in the corresponding response. The Seq field is used in combination with the NetFn and Command fields to form a unique value. I.e. the same Seq value could be used in multiple outstanding requests, as long as the combinations of Seq value, NetFn, and Command were unique among the requests.

Cmd Command code. This message byte specifies the operation that is to be executed under the specified Network Function.

Data Zero or more bytes of data, as required by the given command. The general convention is to pass data LS-byte first, but check the individual command specifications to be sure.

11.2 BMC-BT Interface Response Message Format

Response Messages are *read transfers* from the BMC to system software via the BT Interface. Note that with a few exceptions (e.g., Cold Reset command) the BMC always returns response to a request delivered via the BT interface in order to deliver the completion code, regardless of whether the response has data in the Data field. The message bytes are organized according to the following format specification:

Figure 11-2, BT Interface/BMC Response Message Format

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5:N	Byte 6:N
Length	NetFn/LUN	Seq	Cmd	Completion Code	Data

Where:

Length This is not actually part of the message, but part of the *framing* for the BT Interface. This value is the 1-based count of message bytes following the length byte. The minimum length byte value for a response from the BMC would be 4 to cover the NetFn/LUN, Seq, Cmd, and Completion Code bytes.

LUN Logical Unit Number. This is a return of the LUN that was passed in the Request Message.

NetFn Network Function. This is a return of the NetFn code that was passed in the Request Message.

Seq Used for matching responses up with requests. The BT interface can support interleaved 'multi-threaded' communications. There can be multiple simultaneous outstanding requests from SMS with responses returned asynchronously (and in any order). The Requester (SMS) sets the value for this field. The Responder returns the value in the corresponding response. The Seq field is used in combination with the NetFn and Command fields to form a unique value. I.e. the same Seq value could be used in multiple outstanding requests, as long as the combinations of Seq value, NetFn, and Command were unique among the requests.

Cmd Command. This is a return of the Cmd code that was passed in the Request Message.

Completion Code The Completion Code indicates whether the request completed successfully or not.

Data Zero or more bytes of data. The BMC always returns a response to acknowledge the request, regardless of whether data is returned or not.

11.3 Using the Seq Field

System Management Software is expected to use the Seq field in the following manner. SMS maintains a list of the outstanding requests it has sent. This list holds the Seq, NetFn, and Command values that were used to send the request. There should be one entry in the list for each possible simultaneous outstanding request. When SMS generates a Seq value for a new request, it must ensure that the combination of Seq, Command, and NetFn values do not match any entries already in the outstanding request list.

When a response is received from the BMC, SMS looks for a match between the Seq value, Command, and NetFn values in the response and an entry in the outstanding request list. If there is a match, the response is processed

normally and the outstanding request list entry freed for a new request. If the response does not match, the response can be ignored or passed on to error tracking procedures.

11.4 Response Expiration Handling

It is possible that conditions could occur where a response will not return for a given request. The Seq number associated with the request must be freed so it can be reused. To support this, SMS should implement a response expiration interval.

The BMC must return a response within the specified response time seconds (per the *Get BT Interface Capabilities* command). If the response is not received in this time the corresponding entry in the SMS outstanding response list can be cleared. If retries are not recommended at the interface, a missing response constitutes an immediate error condition. If the interface recommends retries (per the *Get BT Interface Capabilities* command) SMS should retry the request up to the specified count. If the response is still not provided, an error has occurred.

> *The typical BT Interface is expected to be fundamentally reliable without retries. The retry specification is to support possible commands within the controller that may occasionally exceed the Request-to-Response specification. An application can elect to implement retry counts that exceed the recommendation.*

The BMC must not return a given response once the corresponding Request-to-Response interval has passed. The BMC can ensure this by maintaining its own internal list of outstanding requests through the interface. The BMC could age and expire the entries in the list by expiring the entries at an interval that is somewhat shorter than the specified Request-to-Response interval. The BMC can define its own internal Seq value or tracking number for this purpose, or it could use the Seq, NetFn, and Command values in the same manner as SMS.

11.5 Logging Events from System Software via BT Interface

The BT Interface can be used for sending Event Messages from system software to the BMC Event Receiver. The following figures show the format for BT Interface Event Request and corresponding Event Response messages. Note that only Event Request Messages to the BMC via the BT Interface have a Software ID field. This is so the Software ID can be saved in the logged event.

Figure 11-3, BT Interface Event Request Message Format

Length	NetFn (04h = Sensor/Event Request)		LUN (00b)	Seq	Command (02h = Platform Event)	Software ID (Gen ID) 7-bits	1
EvMRev	Sensor Type	Sensor #	Event Dir	Event Type	Event Data 1	Event Data 2	Event Data 3

Shading designates fields that are not stored in the event record.

Figure 11-4, BT Interface Event Response Message Format

Length	NetFn (05h = Sensor/Event Response)	00	Seq	Command (02h = Platform Event)	Completion Code

11.6 Host to BMC Interface

The Host interface to the baseboard management controller (BMC) requires a block of 3 contiguous I/O locations on the system board. (A reference implementation fixes this at locations **E4h:E6h**. The interface circuitry will decode the lower 2 address lines, SA[1..0]). A general-purpose chip select will be used to generate the select line for the interface, which is to reside in system I/O space. The I/O address offsets are defined as follows:

Table 11-1, BT Interface Registers

Offset	Read	Write
0	BT_CTRL - control register	
1	BMC2HOST buffer	HOST2BMC buffer
2	BT_INTMASK - interrupt mask register	

The two buffers must meet the specified maximum message size requirements for all protocols supported on the messaging channels implemented on the BMC. Implementations can choose to provide more depth optionally. The GET_BT_INTERFACE_CAPABILITIES command is used to query for the actual implementation buffer depth.

The messaging protocol involves the host writing the command stream to the BT buffer, followed by setting a "attention" bit in the BT control register. This automatically generates an interrupt to the baseboard management controller (BMC). The BMC then reads command packet from the BT buffer, and clears the attention bit. After processing the command, the BMC then writes the response data to the host-bound buffer. Finally, the BMC sets an outbound attention bit and generates an interrupt to the host (the host may optionally poll the attention bits, and may enable/disable the interrupts via a MASK register). Refer to *Section 11.7* for a walk-through of the sequence of operations used for transfers on the BT interface.

There is no explicit requirement or recommendation for the hardware used to implement the interface. A discrete, custom, programmable array, or other implementation may be used at the discretion of the designer. As an example, some implementations have used a Xilinx* XC4003E Field Programmable Gate Array (FPGA) to implement the interface circuit because it provides on-chip user RAM that can be effectively used to implement the interface's buffers. This implementation was able to provide 64-byte buffers.

11.6.1 BT Host Interface Registers

The Host BT interface provides an independent set of registers and interrupts to allow the Host driver to communicate with the baseboard management controller without conflicting with the O/S ACPI driver.

11.6.2 BT BMC to Host Buffer (BMC2HOST)

From the host side, this is a read-only buffer, which contains a command response stream from the embedded controller. The buffer must be a minimum of 64-bytes deep. This shares offset 1 of the I/O space with the HOST2BMC buffer. Hence I/O read cycles from the host CPU remove data from this buffer, whereas write cycles from the BMC load data into this buffer.

11.6.3 BT Host to BMC Buffer (HOST2BMC)

From the host side, this is a write-only buffer to which the host writes a command stream to the baseboard management controller. The buffer must be a minimum of 64-bytes deep. This shares offset 1 of the I/O space with the BMC2HOST buffer. Hence an I/O write cycles from the host CPU load data into this buffer, whereas read cycles from the BMC remove data from this buffer.

11.6.4 BT Control Register (BT_CTRL)

The host and the BMC use this register for various control functions defined below.

Figure 11-5, BT_CTRL Register format

7	6	5	4	3	2	1	0
B_BUSY	H_BUSY	OEM0	EVT_ATN	B2H_ATN	H2B_ATN	CLR_RD_PTR	CLR_WR_PTR

Table 11-2, BT_CTRL Register Bit Definitions

BIT	R/W* By Host	R/W* By BMC	NAME	FUNCTION
0	W	W	CLR_WR_PTR	Clear Write Pointer. The host writes a 1 to clear the write pointer to the BT *HOST2BMC* buffer; this bit is always read back as 0. Writing a 0 has no effect. Similarly, the BMC writes a 1 to clear the write pointer to the BT *BMC2HOST* buffer; this bit is always read back as 0. Writing a 0 has no effect. Clearing the pointer is defined as moving it to point to the start of the next valid buffer (typically the top of a single FIFO buffer).
1	W	W	CLR_RD_PTR	Clear Read Pointer. The host writes a 1 to clear the read pointer to the BT *BMC2HOST* buffer; this bit is always read back as 0. Writing a 0 has no effect. Similarly, the BMC writes a 1 to clear the read pointer to the BT *HOST2BMC* buffer; this bit is always read back as 0. Writing a 0 has no effect. Clearing the pointer is defined as moving it to point to the start of the next valid buffer (typically the top of a single FIFO buffer).
2	R/S Write 1 to set bit; 0 no effect	R/C Write 1 to clear bit; 0 no effect	H2B_ATN Reset State=0	Host to BMC Attention. When the host writes a 1 to this bit, an interrupt is generated to the baseboard management controller. The host should set this bit when it has completed writing a message stream to the HOST2BMC buffer. The baseboard management controller clears this bit after it has set the B_BUSY bit. The host may poll the H2B_ATN bit to determine that the baseboard management controller has acknowledged the command. The capability to operate in a polled mode by the BMC is optional.
3	R/C Write 1 to clear bit; 0 no effect	R/S Write 1 to set bit; 0 no effect	B2H_ATN Reset State=0	BMC to Host Attention. The BMC sets this bit when it has completed writing a message response stream to the BMC2HOST buffer. The host may poll the B2H_ATN bit to determine that the baseboard management controller has finished writing a message response stream to the BMC2HOST buffer. After setting H_BUSY, the host should clear this bit to acknowledge receipt of the message response. This bit can be enabled to generate an interrupt to the host by setting the B2HI_EN bit in the INTMASK register.
4	R/C Write 1 to clear bit: 0 no effect	R/S Write 1 to set bit; 0 no effect	SMS_ATN Reset State=0	SMS Attention. The BMC sets this bit when it has detected and queued an SMS message that must be reported to the host. This allows the host to distinguish between command responses and SMS messages from the baseboard management controller. This bit can be enabled to generate an interrupt to the host by a host set of the B2HI_EN bit in the INTMASK register. The host clears this bit by writing a 1 to it.

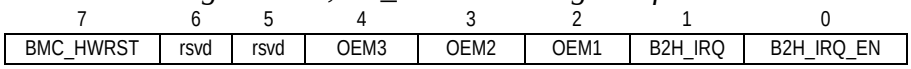
BIT	R/W* By Host	R/W* By BMC	NAME	FUNCTION
5	R/S Write 1 to set bit; 0 no effect	R/C Write 1 to clear bit: 0 no effect	OEM0 Reset State=0	Reserved for definition by platform. Generic IPMI software must write this bit as 0, and ignore the value on read. The OEM0 bit should be able to generate an interrupt to the BMC when written by the host but is not required (polled mode is acceptable). Typical usage is a “heartbeat” mechanism from/to the host; the host sets OEM0 to interrupt the BMC and then polls this bit to be cleared (BMC is alive and responded to the interrupt). The BMC FW completes the acknowledge cycle by clearing OEM0 upon receipt of the interrupt (host is alive).
6	R/S/C Write 1 to toggle	R	H_BUSY Reset State=0	Host Busy. This bit is set/cleared by the Host to indicate that it is busy processing response/event data from the BMC or cannot accept response/event data at this time. It is set to 1 if the host writes a 1 when H_BUSY=0, cleared if the host writes a 1 when H_BUSY=1; there is no effect if the host writes a 0 to this bit (toggle implementation). The BMC will need to verify that this bit is cleared before sending a response or event message.
7	R	R/S/C Write 1 to toggle	B_BUSY Reset State=1	Baseboard Management Controller Busy. This bit is set/cleared by the BMC to indicate that it is busy processing command/request data from the Host or cannot accept command/request data at this time. It is set to 1 if the BMC writes 1 when B_BUSY=0, cleared if the BMC writes 1 when B_BUSY=1; there is no effect if the BMC writes 0 to this bit (toggle implementation). . The initial state of this bit should be set to 1 so that the BMC side driver can initialize and prepare to accept Host traffic before the Host attempts to use it the first time.

* R=read; W=write; S=set; C=clear

11.6.5 BT Interrupt Mask Register (INTMASK)

This register is used by the host to control which interrupts can be generated by the baseboard management controller.

Figure 11-6, BT_INTMASK Register format



7	6	5	4	3	2	1	0
BMC_HWRST	rsvd	rsvd	OEM3	OEM2	OEM1	B2H_IRQ	B2H_IRQ_EN

Table 11-3, BT_INTMASK Register Bit Definitions

BIT	R/W	NAME	FUNCTION
0	R/W	B2H_IRQ_EN	BMC to HOST Interrupt Enable. The interrupt is generated by the BMC-BT interface if B2H_IRQ_EN is set (1) and either the B2H_ATN or EVT_ATN bits are set by the BMC.
1	R/W	B2H_IRQ	BMC to HOST Interrupt Active. This bit reflects the state of the interrupt line to the host, and therefore can only become set (1) if by B2H_IRQ_EN is set and the interrupt condition has occurred. On a read: 0 = interrupt to host not active; 1 = interrupt to host active On a write: 0 = no effect; 1 = clear interrupt (this is the source of the INT, and is immediately cleared by the O/S driver). *This only clears the interrupt for the system interface. Other interrupts may require clearing flags internal to the BMC.* If bit is 0, then a rising edge on B2H_ATN or EVT_ATN sets this to 1. If already 1, then no affect.
2	R/W	OEM1	Reserved for definition by platform manufacturer for BIOS/SMI Handler use. Generic IPMI software must write this bit as 0, and ignore the value on read.
3	R/W	OEM2	Reserved for definition by platform manufacturer for BIOS/SMI Handler use. Generic IPMI software must write this bit as 0, and ignore the value on read.
4	R/W	OEM3	Reserved for definition by platform manufacturer for BIOS/SMI Handler use. Generic IPMI software must write this bit as 0, and ignore the value on read.
5	R/W	Reserved	Reserved for future definition by IPMI. Write as 0, ignore value on read.
6	R/W	Reserved	Reserved for future definition by IPMI. Write as 0, ignore value on read.
7	R/W	BMC_HWRST	Host to Baseboard Management Controller Reset. (OPTIONAL) Always read back as zero. Writing a 1 to this bit will cause a hardware reset of the BMC. This is non-sticky; writing zero has no effect. This bit, if provided, is intended for to be used for error recovery by the host if loss of communication with the BMC occurs.

11.7 Communication Protocol

In the context of the BT Interface, the term *Write Transfer* refers to the Host writing data to the BMC, while *Read Transfer* refers to the Host reading data from the BMC.

If the interface implementation supports multithreaded operation, the interface driver should always be looking for the B2H_ATN or EVT_ATN condition. In an interrupt driven implementation, this means the interrupt handler should always check for responses or asynchronous requests. In a polled implementation, the driver should periodically poll the state of these bits.

Table 11-4, BT Interface Write Transfer

Operation	Host	BMC	H2B_ ATN	B2H_ ATN	B_BUSY	H_BUSY
Start		Enable host interface (Clear B_BUSY)	0	0	1	0
"Command" (Write Transfer)	Wait for B_BUSY clear (BMC ready to accept a request) & H2B_ATN clear (signifying acknowledge of previous command)	Wait for H2B_ATN (indicating data has been loaded into HOST2BMC buffer)	0	0	0	0
	Write 1 to CLR_WR_ PTR bit in BT_CNTRL (reset pointer to start of buffer)	"	0	0	0	0
	Write *bytes 1 to n* of command (request) to HOST2BMC buffer	"	0	0	0	0
	Set H2B_ATN attention (tell BMC that write data is available)	"	1	0	0	0
		Set B_BUSY (indicating BMC is preparing to transfer data from the HOST2BMC buffer)	1	0	1	0
		Clear H2B_ATN (the ACK)	0	0	1	0
		Read HOST2BMC buffer	0	0	1	0
		Clear B_BUSY (indicating BMC is done transferring data)	0	0	0	0
		Process command	0	0	0	0

Table 11-5, BT Interface Read Transfer

Operation	Host	BMC	H2B_ ATN	B2H_ ATN	B_ BUSY	H_BUSY
"Response" (Read Transfer)	Wait for B2H_ATN attention to be set (or wait for interrupt from BMC), signaling BMC has data available for Host.	Waits for H_BUSY to be cleared	0	0	0	0
	"	Write bytes 1 to n of response to BMC2HOST buffer	0	0	0	0
	"	Set B2H_ATN (indicating BMC has put data in BMC2HOST buffer)	0	1	0	0
	Set H_BUSY (indicating Host is in process of reading data from the interface)	Wait for B2H_ATN clear (ACK of BMC response message)	0	1	0	1
	Clear B2H_ATN	"	0	0	0	1
	Write 1 to CLR_RD_PTR bit in BT_CTRL	"	0	0	0	1
	Read *bytes 1 to n* of response phase from BMC2HOST buffer	"	0	0	0	1
	Clear H_BUSY (indicating Host has completed reading data from the buffer)	"	0	0	0	0
Idle		"	0	0	0	0

11.8 Host and BMC Busy States

The host and BMC can set H_BUSY and B_BUSY, respectively, as necessary to indicate they are not able to accept response/event or command data from the BMC or host, respectively for any reason. This allows for asynchronous housekeeping functions that might take an extended period of time (seconds or minutes) to be accomplished in a controlled manner and minimize the chance of getting out of synchronization - which might occur if the host or BMC "timed out" and assumed the other side was hung or not responding.

11.9 Host Command Power-On/Reset States

The BMC sets B_BUSY to 1 whenever it is initializing from a cold reset and following BMC power up. The interface will initialize with H_BUSY, H2B_ATN, and B2H_ATN set to 0 (reset state = 0).

12. IPMI LAN Interface

This section describes the mechanisms specific to transferring IPMI messages between the BMC and a remote management system (remote console) over an Ethernet LAN connection using UDP under IPv4. The UDP datagrams are formatted to contain IPMI request and response messages, plus additional messages for discovery and authentication.

While an IPMI LAN interface can be accomplished using a LAN Controller that is dedicated to the BMC, it will usually be accomplished using LAN Controller that can be shared for both BMC and system use.

There are two implementations that are likely to be used to deploy an IPMI LAN Interface using a shared LAN controller. The first implementation is using an embedded LAN controller, as shown in *Figure 12-1*, and the second is using a LAN controller on an add-in card, as shown in *Figure 12-2*.

Both examples show a LAN Controller that has the capability to detect UDP datagrams sent to a ‘management port’. Any datagrams received on that port are forwarded to a ‘side-band’ interface that allows them to be delivered to, or retrieved by, the BMC. As *Figure 12-1* shows, these incoming ‘platform management’ datagrams may also be delivered to system software in parallel with being delivered to the BMC.

The BMC can use this same interface to inject datagrams onto the LAN. These datagrams are interleaved with the network packets that are generated by system software.

The LAN Controller can be designed in such a way that the interface for the ‘management port’ is powered by standby power and remains operative even when the system is powered down. This provides a mechanism that allows IPMI LAN messaging to occur independent from system software and the system’s power state. A LAN controller dedicated to the BMC can also be used.

Figure 12-1, Embedded LAN Controller Implementation

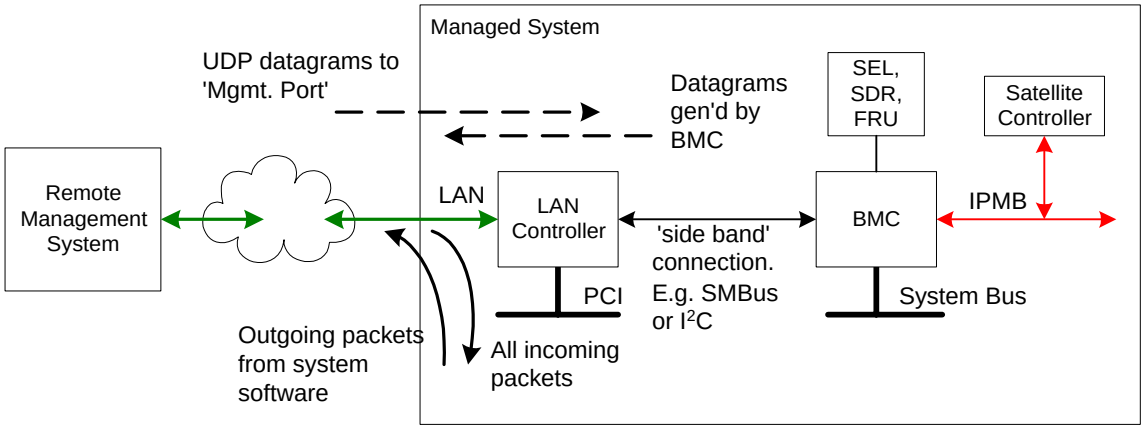
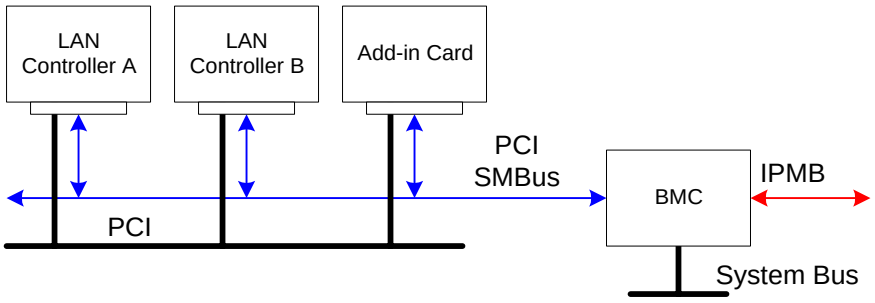


Figure 12-2 shows an implementation where the LAN Controller is implemented as a PCI add-in card connected to the BMC via a PCI Management Bus connection. This approach avoids the need to have the LAN Controller built-into the system, allowing the LAN Controller portion of the IPMI LAN Interface to be added or updated at a later time.

Figure 12-2, PCI Management Bus Implementation



12.1 RMCP

The Distributed Management Task Force (DMTF) has defined a 'Remote Management Control Protocol' (RMCP) for supporting pre-OS and OS-absent management. RMCP is a simple request-response protocol that can be delivered using UDP datagrams. IPMI-over-LAN uses version 1 of the RMCP protocol and packet format.

RMCP includes a field that indicates the class of messages that can be embedded in an RMCP message packet, including a class for IPMI messages. Other message classes are 'ASF' and 'OEM'.

IPMI LAN messages are encapsulated in RMCP packets using the IPMI message class. An IPMI LAN implementation can also use ASF-class 'Ping' and 'Pong' messages to support the discovery of IPMI managed systems on the network.

12.1.1 ASF Messages in RMCP

The term 'ASF' is commonly used in RMCP. ASF originally stood for 'Alerting Standard Forum'. This is the original name of a group that has moved into the DMTF as the Pre-OS Working Group. The group is standardizing a set of messages under RMCP that are oriented towards non-intelligent management hardware supporting basic LAN Alerting and recovery control (e.g. system reset) capabilities via the LAN.

RMCP uses 'ASF' to denote the fields and values that support these messages.

12.1.2 RMCP Port Numbers

RMCP uses two well-known ports under UDP. The following table describes these ports and summarizes their use.

Table 12-1, RMCP Port Numbers

Port #	Name	Description
623 (26Fh)	Aux Bus Shunt (Primary RMCP Port)	Hereon referred to as the Primary RMCP Port - This port and the required RMCP messages must be provided to be conformant with the RMCP specifications. There is a mandatory set of messages that are required to be supported on this port. These messages are always sent 'in the clear' so that system software can discover systems that have RMCP support.
664 (298h)	Secure Aux Bus (Secondary RMCP Port)	Hereon referred to as the Secondary RMCP Port or Secure Port. This port is only used when it is necessary to encrypt packets using an algorithm or specification that prevents also sending unencrypted packets from being transferred via the same port. Since discovery requires sending 'in the clear' RMCP Ping/Pong packets, the secondary port is used to transfer encrypted transfers while the primary port continues to support unencrypted packets. An implementation that utilizes this port must still support the Primary RMCP Port and the required messages on that port in order to be conformant with the RMCP specifications. Note that the common IPMI messaging protocols and authentication mechanisms in this specification do not use encrypted packets, therefore IPMI messaging does not need to use the secondary port.

12.1.3 RMCP Message Format

There are two types of RMCP messages: Data or 'Normal' RMCP messages, and RMCP Acknowledge Messages. Data messages and ACK messages are differentiated by the ACK/normal bit of the Class of Message field.

Table 12-2, RMCP Message Format

Field	size in bytes	Description
RMCP Header		
Version	1	06h = RMCP Version 1.0
Reserved	1	00h
Sequence Number	1	varies, see text
Class of Message	1	This field identifies the format of the messages that follow this header. All messages of class ASF (6) conform to the formats defined in this specification and can be extended via an OEM IANA. Bit 7 RMCP ACK 0 - Normal RMCP message 1 - RMCP ACK message Bit 6:5 Reserved Bit 4:0 Message Class 0-5 = Reserved 6 = ASF 7 = IPMI 8 = OEM defined all other = Reserved
RMCP Data		
Data	Variable	data based class of message

The following table presents how the ACK/Normal Bit and the Message Class combine to identify the type of message under RMCP and which specification defines the format of the associated message data.

Table 12-3, Message Type Determination Under RMCP

ACK/Normal bit	Message Class	Message Type	Message Data
ACK	ASF	RMCP ACK	No Data. Message just contains RMCP Header with the Sequence Number set to the sequence number from the last message that was received.
ACK	all other	undefined	not allowed
normal	ASF	ASF Messages	Per ASF Specification
normal	OEM	OEM Message under RMCP	bytes 0:3 = OEM IANA
			bytes 4:N = OEM Message Data (defined by manufacturer or organization identified by the OEM IANA field value)
normal	IPMI	IPMI Messages	Per this specification

12.2 Required ASF/RMCP Messages for IPMI-over-LAN

The following class=ASF messages under RMCP must be supported in a system implementing the IPMI LAN interfaces over TCP/IP-UDP. This is just a specification of the minimum ASF message support required for IPMI LAN implementations. IPMI LAN messaging can coexist with additional ASF messaging on a system. Therefore, a system can support additional ASF messages and functions without being non-conformant with the IPMI LAN specifications.

There is no IPMI requirement for the BMC to respond to RMCP Messages of class=ASF other than the RMCP Ping message. However, additional message support may be required if the system is also to be conformant with the ASF specification. Refer to [ASF].

Table 12-4, ASF/RMCP Messages for IPMI-over-LAN

Message	Description
RMCP ACK	RECOMMENDED. Per the ASF specifications the RMCP ACK message should be returned whenever a 'normal' RMCP message with an RMCP sequence number of 0-254 is received. This is recommended, but not required for a system to be conformant with the IPMI LAN specification. (Note, however, that a system that does not return the ACK is not fully conformant with the RMCP specification). See sections *12.2.1, RMCP ACK Messages*, and *12.2.2, RMCP ACK Handling* for more information.
ASF Presence Ping message	REQUIRED. This message returns information about the interfaces supported via RMCP. It is used both to discover managed systems that support RMCP and to determine whether the system supports IPMI LAN messaging and/or additional ASF commands. This message must be supported on the Primary RMCP port.
ASF Presence Pong Message (Ping response)	REQUIRED. This message must be returned from the managed system in response to the Presence Ping message on the Primary RMCP port.

12.2.1 RMCP ACK Messages

Table 12-5, RMCP ACK Message Fields, shows the RMCP header and data values for the RMCP ACK message. This message is used to acknowledge receipt of a 'normal' RMCP messages that were transmitted with a 0-254 RMCP sequence number. RMCP ACK messages are not generated if the RMCP sequence number is 255 (FFh). The RMCP ACK message does not indicate that an action has been completed, only that a specific RMCP packet has been received.

The RMCP ACK operation is defined as being symmetric. That is, any party that receives a normal RMCP message with a 0-254 RMCP sequence number is supposed to respond with an RMCP ACK message. Thus, RMCP ACK messages can be generated by remote consoles and managed systems.

Table 12-5, RMCP ACK Message Fields

Field	Value
Version	Copied from received message.
Reserved	Copied from received message.
Sequence Number	Copied from received message.
Class of Message	7 Set to 1 to indicate 'ACK' packet 6:0 Copied from received message.
RMCP Data	none

12.2.2 RMCP ACK Handling

RMCP ACK messages are not required for IPMI messaging, since IPMI already has its own messaging retry policies. In addition, some Network Controllers usable for IPMI messaging do not automatically generate

RMCP ACK messages. In these implementations, the BMC would have to generate the RMCP ACK, resulting in additional, unnecessary traffic from the BMC. Therefore, RMCP ACK messages should not be used for IPMI messaging. This leads to the following requirements and recommendations:

- RMCP messages with class=IPMI must have their RMCP sequence number set to 255 (FFh) to indicate that RMCP ACK messages are not to be generated by the message receiver.
- Console software should also set the RMCP sequence number to 255 (FFh) for non-IPMI messages, whenever possible. Some systems may not respond with an RMCP ACK for non-IPMI messages even if one was requested using a 0-254 RMCP sequence number. Console software should be prepared for this occurrence. The software can discover which systems support RMCP ACK by checking to see whether RMCP ACKs are generated as the result of sending RMCP *Presence Ping* messages. If RMCP ACKs are not received, the software should proceed without requiring RMCP ACK messages.
- Regardless of whether RMCP ACK messages are received from a system, console software should still send RMCP ACKs whenever it receives an RMCP message with a 0-254 RMCP sequence number.

12.2.3 RMCP/ASF Presence Ping Message

This message returns information about the interfaces supported via RMCP. It is used both to discover managed systems that support RMCP and to determine whether the system supports IPMI LAN messaging and/or additional ASF commands. The following table illustrates the specific fields to be used for Presence Ping Message to a system implementing IPMI LAN messaging.

Table 12-6, RMCP Packet Fields for ASF Presence Ping Message (Ping Request)

	Field	size in bytes	Value
UDP Header	Source Port	2	per UDP
	Destination Port	2	26Fh
	UDP Length	2	per UDP
	UDP Checksum	2	per UDP
RMCP Header	Version	1	06h = RMCP Version 1.0
	Reserved	1	00h
	RMCP Sequence Number	1	0-254 if RMCP ACK desired. 255 for no RMCP ACK. See sections *12.2.1, RMCP ACK Messages*, and *12.2.2, RMCP ACK Handling* for more information.[1]
	Class of Message	1	06h for ASF
ASF Message	IANA Enterprise Number	4	4542 (ASF IANA)
	Message Type	1	80h = Presence Ping
	Message Tag	1	0-FEh, generated by remote console. This is an RMCP version of a sequence number. Values 0-254 (0-FEh) are used for RMCP request/response messages. 255 indicates the message is unidirectional and not part of a request/response pair.
	Reserved	1	00h
	Data Length	1	00h

1. Some systems may not generate RMCP ACKs even if requested. Software should be designed to handle this occurrence.

12.2.4 RMCP/ASF Pong Message (Ping Response)

This message must be returned from the managed system in response to the Presence Ping message.

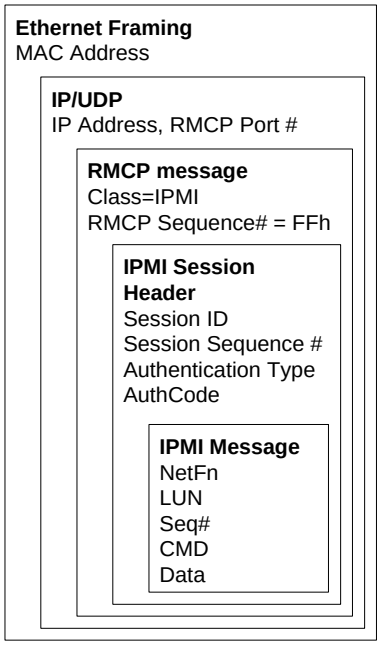
Table 12-7, RMCP Packet Fields for ASF Presence Pong Message (Ping Response)

	Field	size in bytes	Value
UDP Header	Source Port	2	26Fh
	Destination Port	2	from Ping request
	UDP Length	2	per UDP
	UDP Checksum	2	per UDP
RMCP Header	Version	1	6 = RMCP Version 1.0
	Reserved	1	00h
	RMCP Sequence Number	1	FFh for IPMI[2]
	Class of Message	1	06h = ASF
ASF Message	IANA Enterprise Number	4	4542 = ASF IANA
	Message Type	1	40h = Presence Pong
	Message Tag	1	from Ping request
	Reserved	1	00h
	Data Length	1	16 (10h)
	IANA Enterprise Number	4	If no OEM-specific capabilities exist, this field contains the ASF IANA (4542) and the OEM-defined field is set to all zeroes (00000000h). Otherwise, this field contains the OEM's IANA Enterprise Number and the OEM-defined field contains the OEM-specific capabilities.
	OEM-defined	4	Not used for IPMI. This field can contain OEM-defined values; the definition of these values is left to the manufacturer identified by the preceding IANA Enterprise number.
	Supported Entities	1	81h for IPMI [7] 1b = IPMI Supported [6:4] Reserved [3:0] 0001b = ASF Version 1.0
	Supported Interactions[1]	1	Reserved for future definition by ASF specification, set to 00000000b
	Reserved	6	Reserved for future definition by ASF specification, set to 00 00 00 00 00 00h

12.3 IPMI Messages Encapsulation Under RMCP

For LAN transfers, IPMI messages are a special class of data encapsulated in an IPMI Session packet. The IPMI Session packets are encapsulated in RMCP packets, which are encapsulated in UDP datagrams. This is illustrated in the following figure. The same type of encapsulation is used for IPMI serial/modem messages via PPP, except the Ethernet Framing is replaced with a packet that uses PPP Framing and IP protocol type.

Figure 12-3, IPMI LAN Packet Layering



12.3.1 RMCP/ASF and IPMI Byte Order

Please take note of the following:

Multi-byte fields in RMCP/ASF fields are specified as being transmitted in 'Network Byte Order' - meaning most-significant byte first.
RMCP and ASF-specified fields are therefore transferred most-significant byte first.

The IPMI convention is to transfer multi-byte numeric fields least-significant Byte first. Therefore, unless otherwise specified:
Data in the IPMI Session Header and IPMI Message fields are transmitted least-significant byte first.

12.3.2 Example IPMI over LAN Packet

The following table shows the format and fields for IPMI messages encapsulated in an RMCP packet that is itself encapsulated within an IPv4 UDP packet delivered over Ethernet:

Table 12-8, RMCP Packet for IPMI via Ethernet

	Field	size in bytes	Byte Offset	Value
MAC Header	Destination Address	6	00h	
	Source Address	6	06h	
	Frame Type	2	0Ch	0800h
IP Header	Version	4-bits	0Eh	4h for IPv4
	Header Length (length of IP header in units of 4-bytes)	4-bits	0Eh	5h
	Precedence	3-bits	0Fh	000b[4]
	Service Type (Type of Service)	4-bits	0Fh	1000b[4] (minimize delay)
	reserved	1-bit	0Fh	0b
	Total Length	2	10h	
	Identification[5]	2	12h	note[5]
	Flags	3-bits	14h	010b[6] (don't fragment)
	Fragment Offset	13-bits	14h	0_0000_0000_0000b[7]
	Time-to-Live	1	16h	40h[3]
	Protocol	1	17h	11h
	Header Checksum	2	18h	
	Source IP Address	4	1Ah	
	Destination IP Address	4	1Eh	
UDP Header	Source Port	2	22h	
	Destination Port	2	24h	26Fh
	UDP Length	2	26h	
	UDP Checksum	2	28h	
RMCP Header	Version	1	2Ah	
	Reserved	1	2Bh	
	RMCP Sequence Number	1	2Ch	FFh for IPMI[2]
	Class of Message	1	2Dh	07h for IPMI
IPMI Session	Authentication Type	1	2Eh	
	Session Sequence #	4	2Fh	note[8]
	Session ID	4	33h	note[8]
	Message Authentication Code (AuthCode) This field is not present when Authentication Type set to 'none'.	16	37h	
	IPMI Message Length	1	47h	
IPMI Message	Per *Section 12.4, IPMI LAN Message Format*	varies	48h:xx	
	PAD[1]	1		
MAC level	CRC	4		

1. Some LAN adapter chips may have a problem where packets of overall lengths 56, 84, 112, 128, or 156 are not handled correctly. The PAD byte is added as necessary to avoid these overall lengths. Remote console software must use the PAD byte when formatting packets to any 10/100 Ethernet device that accepts RMCP packets.
2. RMCP Messages with class=IPMI should be sent with an RMCP Sequence Number of FFh to indicate that an RMCP ACK message should not be generated by the message receiver.

3. Default value for packets transmitted from the BMC. Can be overridden via a configuration parameter setting.
4. Value used for packets transmitted from the BMC. The BMC ignores the value of this parameter (except for checksum calculations) on received packets.
5. BMC should increment this field each time it sends a new packet.
6. Default value for packets transmitted from the BMC. Bit offset 1 (fragment bit) can be overridden via a configuration parameter setting.
7. Default value for packets transmitted from the BMC. The BMC is not required to support receiving fragmented packets. Packets with a non-zero fragment offset and/or a flags field bit 2 = 1b ("more fragments" may be silently discarded.)
8. The Session ID and Session Sequence Number must be non-zero for commands executed during an active session. All 0's for the Session ID and/or Session Sequence Number (null Session ID, null Session Sequence Number) are special values only used for commands that can be executed prior to establishing a session, e.g. Get System GUID, Get Channel Authentication Capabilities, and Get Session Challenge. The Activate Session uses a null Session Sequence Number before a session is activated, but does not use a null Session ID. Instead, it must use the Temporary Session ID given by the BMC in the response to the Get Session Challenge command.

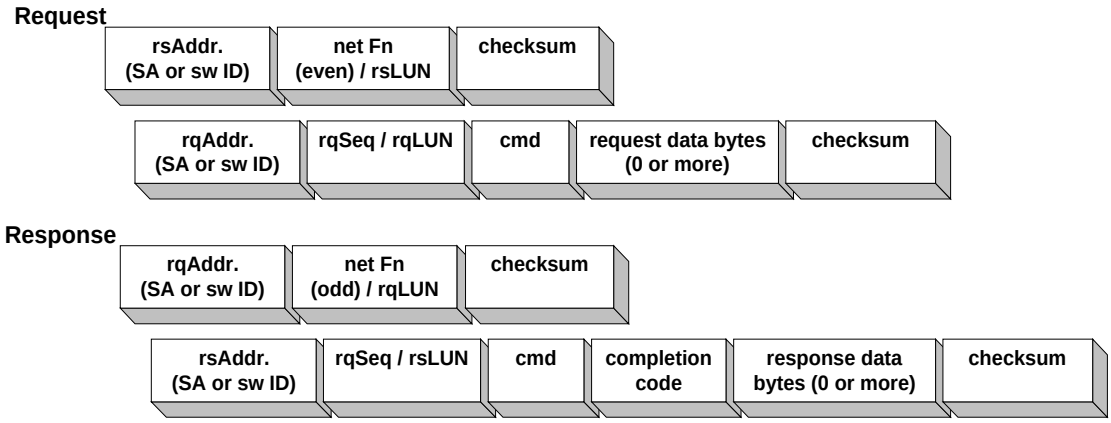
12.4 IPMI LAN Message Format

The encapsulated IPMI Messages are based on the same format as specified for the IPMB. This is done for consistency and simplification of bridging operations. There is one significant difference. For IPMB messages, the requester and responder addresses are always 7-bit I^2C slave addresses. For IPMI LAN messages, the addresses can be either slave addresses or software IDs. The least significant bit of the responder's address and requester's address field indicates which type of address is being used, as described below.

There is no linkage between inbound and outbound messages and whether the message is a request or a response message. Inbound messages can be either request or response messages and outbound messages can be request or response messages.

The following table presents the formats for request and response messages:

Figure 12-4, IPMI LAN Message Formats



Where:

checksum	2's complement checksum of preceding bytes in the connection header or between the previous checksum. 8-bit checksum algorithm: Initialize checksum to 0. For each byte, checksum = (checksum + byte) modulo 256. Then checksum = - checksum. When the checksum and the bytes are added together, modulo 256, the result should be 0.
cmd	Command Byte
completion code	Completion code returned in the response to indicated success/failure status of the request.

data	As required by the particular request or response for the command
LUN	The lower 2-bits of the netFn byte identify the logical unit number, which provides further sub-addressing within the target node.
netFn	Network Function code
rq	Abbreviation for ‘Requester’.
rqLUN	Requester’s LUN.
rqAddr	Requester's Address. 1 byte. LS bit is 0 for Slave Addresses and 1 for Software IDs. Upper 7-bits hold Slave Address or Software ID, respectively. This byte is always 20h when the BMC is the requester.
rqSeq	Sequence number, generated by the requester.
rs	Abbreviation for ‘Responder’.
rsLUN	Responder’s LUN
rsAddr	Responder's Slave Address. 1 byte. LS bit is 0 for Slave Addresses and 1 for Software IDs. Upper 7-bits hold Slave Address or Software ID, respectively. This byte is always 20h when the BMC is the responder.
Seq	Sequence number. This field is used to verify that a response is for a particular instance of a request. Refer to [IPMB] for additional information on use and operation of the Seq field.

12.5 LAN Alerting

LAN Alerts are accomplished by generating a UDP Datagram that contains an SNMP Trap formatted per the IPMI Platform Event Trap (PET) Format specification. This same format is used for PPP alerts generated over the serial/modem interface when operating in PPP/UDP mode. Information for the PET trap comes from the Event Message that generated the alert and from the LAN configuration parameters for PET.

12.6 IPMI LAN Configuration

12.6.1 IP and MAC Address Configuration

The BMC in the managed system needs the system’s IP Address and MAC Address in order to be able to respond to UDP/IP packets or generate LAN alerts.

A BMC implementation is not required to be able to run DHCP or other protocols to keep it’s IP address assignment up-to-date. In such implementations, it is the responsibility of system software to keep this address information current in case it might change (as could be the case if the lease expired on an IP address, perhaps because the system was unplugged for a long time).

It is recommended that system software periodically check the BMC’s address assignment to see if it is current, and to update it if it’s not. It is also recommended that the BIOS run DHCP and initialize the BMC IP address on startup if the BMC implementation does not include built-in DHCP support.

12.6.2 ‘Teamed’ and Fail-over LAN Channels

It is possible that an implementation may have multiple network controllers connected to the BMC. In such a configuration, it may be desirable to support a configuration where multiple network controllers share the same IP address. This ‘teamed’ configuration provides a bandwidth improvement by allowing messages to that IP address to be sent and received by multiple NICs. Similar arrangements can be used to offer ‘fail-over’ capability where one NIC will be activated if another fails.

Teaming and fail-over require special system software and driver support that is outside the scope of this specification. However, it should be noted that IPMI Sessions could be implemented in a manner that can

facilitate such applications. One useful approach is to design the implementation such that Session IDs are unique across all channels. That way, if two LAN channels were configured with the same IP address, the BMC could accept session traffic that was split across the two channels. Since user information is channel-specific, it would also be necessary for the user data and other configuration options to be identically configured. An alternative implementation may provide a proprietary option where the two LAN connections are combined into a single logical channel when teaming is in effect.

> Note: The maximum operating privilege level and authentication will be determined by the user privilege and channel privilege limit settings. Since these can vary on a per channel basis, it is possible that unless the channels are configured identically a different maximum operating privilege level will be seen based on which channel a message is received on.

12.7 ARP Handling and Gratuitous ARP

For Ethernet, the Address Resolution Protocol (ARP) [RFC 826] allows a host to find the physical address (MAC address) of a target host on the same network, given only the target's Internet address. Systems and routers cache IP Address-to-MAC Address information so they do not need to perform ARP requests every time they communicate with another system. This cache is commonly referred to as an ARP Cache.

ARP Requests are broadcast. The request contains the IP Address for which an ARP Response containing the MAC address is desired. The sender's IP Address-to-MAC Address mapping is also in every ARP request broadcast. This allows receivers to update their own caches with the sender's information before responding to the ARP request.

A Gratuitous ARP is an ARP Response where the responder sends out the internet-to-physical mapping of its own IP Address. Since the sender's internet-to-physical address mapping is part the request, receivers use that information to update their own caches with the sender's address mapping.

It is common for systems to do a Gratuitous ARP on startup to inform other machines of its address (possibly a new address). This gives the other systems a chance to update their ARP cache entries immediately. A Gratuitous ARP at startup can also be used as a way to check whether another system is already using the system's IP address.

For this version of the specification, Gratuitous ARP capability is only described for Ethernet LAN channels.

12.7.1 OS-Absent problems with ARP

Some BMC LAN implementations may only have the ability to only receive UDP packets that are addressed to the RMCP ports. Since Ethernet ARP packets are not UDP packets, Ethernet ARP request packets would not get routed to the BMC. Thus, when the system is in a powered down state, the system may not accept ARP Requests, or the request may not be able to be seen by the BMC, and the ARP Request will not get responded to. This means that a remote application that relies on ARP to get the MAC address will not be able to connect to the managed system once the system has powered down or is in a sleep state and the remote application's or intermediate router's ARP cache entries expire.

12.7.2 Resolving ARP issues

The following are possible approaches to eliminating or reducing issues that can occur if the BMC LAN implementation cannot receive or respond to ARP Requests while the system is powered down or sleeping. It is also possible for this to happen if the run-time software does not use the network. This could happen while in a failed state or if the system has booted to 'DOS' or a local diagnostic partition.

- Increase ARP Cache expiration intervals in routers and applications.
- Implement Proxy ARP on the subnet - implement Proxy ARP software (software that responds to ARP Requests on behalf of the managed systems) on one or more systems on the subnet. At least one of the Proxy ARP systems must remain powered up.

- Have the application maintain the ARP table - This only works if the remote console application is on the same subnet as the managed system. Some network stacks include an *arp* utility program that allows ARP entries to be manually entered into the ARP table (cache) for that system. An application could use this mechanism to maintain the ARP table with a 'fixed' IP-to-MAC address association for the system.
- Use a router with Proxy ARP capability - Some routers can be configured to provide a Proxy ARP capability
- Wake-On-LAN - If the managed system supports Wake-On-LAN it may be used to wake the system in order to allow system software to respond to a later ARP Request.
- Use a Network Controller with built-in ARP Response capability. As out-of-band management using RMCP becomes more popular, network controller vendors may offer controllers with the ability to directly respond to ARP Requests when the system is powered down or sleeping.
- Provide Gratuitous ARPs from the BMC. If the BMC LAN connection allows the BMC to send ARP Requests, then the BMC could periodically issue Gratuitous ARPs. Many routers and network stacks will accept this Gratuitous ARP in place of an actual ARP response packet.

The best solution is to have an implementation where the BMC or Network Controller directly responds to ARP Requests during times that the OS does not. If this is not possible, having the BMC issue Gratuitous ARPs can often work well as a substitute. Because BMC-generated Gratuitous ARPs and ARP Responses may be common, this specification includes commands that can be used for configuring and controlling those capabilities if they exist in the implementation.

12.7.3 BMC-generated ARPs

A BMC LAN implementation may support BMC-generated Gratuitous ARPs or BMC-generated ARP responses. If either of these options are supported, the BMC shall also support the *BMC-generated ARP Control* LAN configuration parameter.

The term "BMC-generated" in this case means that the Gratuitous ARP or ARP Response generation is under direct control of the BMC. The actual logic for sending the ARP packet may be in another device. For example, a Network Controller chip may have the ability to be enabled by the BMC through a private interface.

It is possible that run-time software will want to take over the responsibilities for ARP handling during run-time. A BMC implementation that supports BMC-generated ARPs should also support the *Suspend BMC ARPs* command. This command allows system management software to suspend BMC-generated ARPs while the Watchdog Timer is running. Refer *19.3, Suspend BMC ARPs Command* for more information.

12.8 Retaining IP Addresses in a DHCP Environment

DHCP (Dynamic Host Configuration Protocol) is an UDP-based protocol that is primarily used to allow systems to obtain an IP Address from a *DHCP Server* on the network. This address assignment is 'leased' and will expire if the assignment is not refreshed by the time the lease expires. The BMC LAN implementation may not be able to run DHCP. This could be because the BMC LAN implementation may only have the ability to send and receive via the RMCP port addresses, preventing it from running standard DCHP.

If the BMC itself cannot run DHCP, the BMC must rely on the IP Address assignment that is configured into the LAN Configuration Parameters. Typically, system software be able to keep the address assignment while the system is running. This can either occur as a consequence of having sufficient IP traffic activity occur to keep the lease, or if the system may be idle for long periods of time, a software agent could be written that periodically refreshes the assignment.

A more serious issue can occur while the system is powered down or sleeping. If the system is powered down or is sleeping for a sufficiently long time, the IP Address could be lost due to expiration of the DHCP lease. When

the system starts up again, the BMC will need to get a new IP address assignment into its configuration parameters.

12.8.1 Resolving DHCP issues

The following are possible approaches to eliminating or reducing issues that can occur if the BMC LAN implementation cannot perform DHCP while the system is powered down or is sleeping:

- If possible, configure Static IP addresses for your managed server systems. DHCP Servers can typically be configured to deliver fixed IP addresses for a given MAC address.
- If you have to use leased IP addresses, configure long lease intervals for the addresses.
- Have a system management software agent that checks the IP Address assignment and updates the BMC if the assignment changes.
- Have the BIOS perform DHCP and update the BMC when the system powers up or resets. This helps safeguard against changes to the IP address that may have occurred when the system was powered down or sleeping. It also helps ensure that the BMC as an IP Address assignment if booting to an alternate OS or service partition, and provides a mechanism for getting an IP address for BMC LAN even before the system has an OS loaded.
- Enable Wake-On-LAN capabilities. This capability can be used to allow a remote console to occasionally wake the system to ensure that the IP Address assignment is retained or updated.

In general, a system in a DHCP environment will typically be used frequently enough to never lose its address assignment. If run-time software and BIOS can keep the BMC up-to-date with IP address assignment changes, the need to refresh assignments while the system is powered-down or sleeping may not be an issue in many environments.

12.9 LAN Session Activation

The LAN Channel is an authenticated multi-session connection. Messages delivered to the BMC via LAN are optionally authenticated using the session authentication mechanisms and challenge/response protocol described in section *6.11.7, Session Activation and IPMI Challenge-Response*. Also refer to sections *6.9, Users & Password Support*, and *6.11.3, Multi-session Connections*.

In addition, a LAN implementation supports discovery via the RMCP Ping/Pong mechanism as a step that typically precedes the session activation phase.

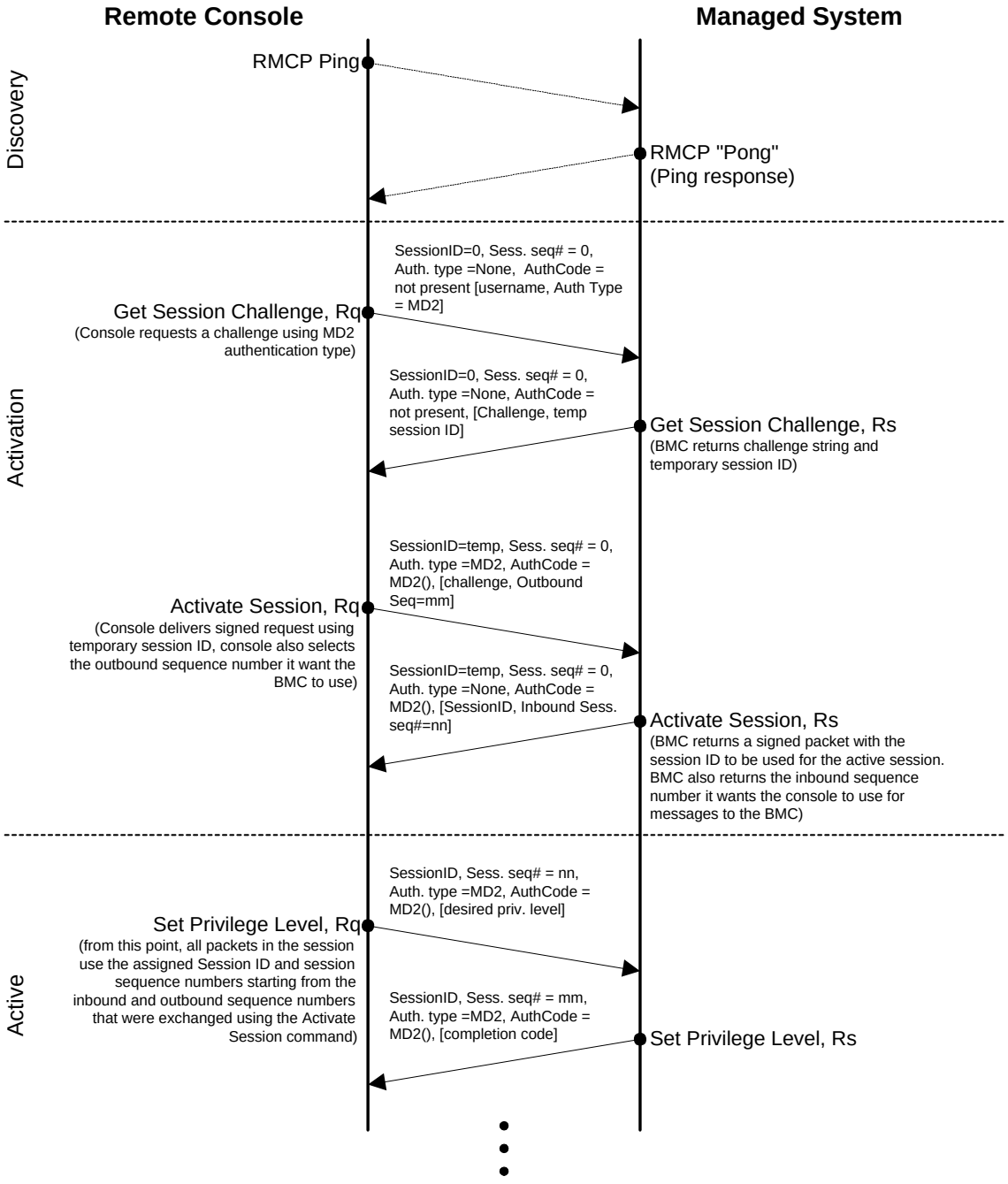
The following presents an overview of the steps that are used by a remote console to establish a IPMI Session via IPMI LAN. These are also illustrated in *Figure 12-5, LAN Session Start*, below.

1. The remote console discovers the system by issuing an RMCP *Presence Ping* message. The response called the *Presence Pong* message, returns a bit indicating whether the platform supports IPMI, and whether the platform uses just the Primary RMCP Port (26Fh) or both the Primary RMCP Port and the Secondary/Secure RMCP Port (298h).
2. If the system supports IPMI, the remote console starts the process of establishing a session by sending a *Get Channel Authentication Capabilities* command packet with Authentication Type = none (“in clear”). The response packet will contain information regarding which type of challenge/response authentication is available to be utilized.
3. The console then requests a session challenge by issuing a *Get Session Challenge* request, also with Authentication Type = none. The request contains information indicating what type of authentication type the console wants to use. This must be one of the supported types returned by the *Get Channel Authentication Capabilities* command. The response packet will contain a challenge string and a Session ID.

4. The console then activates the session by issuing an *Activate Session* request. The Activate Session packet is typically authenticated. For message-digest algorithms, the packet includes a signature (AuthCode) that is a hash of the challenge, the Session ID, the password, and the message data, using one of the supported algorithms from the *Get Channel Authentication Capabilities* command. The console also sets the initial value for the Outbound sequence number that it wants the BMC to use for packets it sends to the console.

5. The BMC returns a response confirming that the Session has been successfully activated. It also returns the Session ID to be used for the remainder of the session, and the initial Inbound session sequence number that it wants the remote console to use for subsequent messages it sends to the BMC for that session. The *Activate Session* response is also authenticated (signed) in the same manner as the request was. This allows the remote console to validate that it has a correct Session ID. Note that IPMI does not support switching authentication algorithms 'mid stream'. The algorithm used with the *Activate Session* command is the algorithm that will be used for subsequent authenticated messages for the session. The exception to this is that the 'none' authentication type is allowed if options such as 'Per-Message Authentication' and/or 'User Authentication' are disabled.

Figure 12-5, LAN Session Startup



13. IPMI Serial/Modem Interface

This section describes the mechanisms specific to transferring IPMI messages between the BMC and a remote management system (remote console) over modem or direct serial connection. It also describes the mechanism that support the Serial Port Sharing capability.

13.1 Serial/Modem Capabilities

The following is a review of the capabilities that can be provided via an IPMI serial/modem connection:

IPMI messaging Transmission of IPMI messages between a remote console and the BMC in one of three configurable modes: Basic Mode, PPP Mode, and Terminal Mode.

Dial Paging Ability to generate a numeric page by sending a dial string to a modem.

TAP Paging Ability to automatically generate a configurable alphanumeric page by automatically connecting to a TAP v1.8 -based paging service.

Dial-out PET Alerting Ability to automatically dial up a remote PPP-to-LAN gateway, connect, and place a Platform Event Trap onto the remote LAN. Also known as PPP Alerting.

Callback Ability for a remote console to trigger the BMC to call the remote console back and establish a system management session. There are two types of Call-back: “IPMI” callback, which is initiated via an IPMI command to the BMC, and callback using Microsoft’s CBCP (callback control protocol). CBCP is an option that is only available in PPP Mode.

PPP UDP Proxy Option to allow the BMC to function as a low-performance communication bridge to allow software to sending and receiving UDP data via a pre-established BMC PPP connection. If the call-back option is supported, local management software or BIOS can trigger the BMC to dial up the remote console.

Serial Port Sharing Ability to share a serial connector between the BMC’s serial controller and a system serial controller by using circuitry to allow it to be switched between the two.

13.2 Connection Modes

The specification for the serial/modem interface supports IPMI Messaging in three possible connection modes. Support for Basic Mode is mandatory if serial/modem support is provided. A given implementation can implement any number or combination of the other connection mode options.

Basic Mode This mode uses a simple clear text password to activate a session. IPMI messages are encoded and delimited using a simple framing scheme based on ‘escaped’ characters. Basic Mode is the most efficient standard operating mode for enabling a remote console application to communicate with the BMC using IPMI messages.

PPP/UDP Mode This mode uses the same session and authentication operation as IPMI over LAN. It uses PPP as the protocol for establishing a point-to-point communications link over which IPMI messages are sent encapsulated in UDP datagrams. This mode incurs significant overhead in message size and handshake complexity beyond that required for Basic Mode IPMI messaging, but has the advantage of using a widely supported standard.

Terminal Mode This mode is intended primarily for direct serial connection operation. The mode is designed so that a simple terminal or terminal emulator can be used to generate requests and get responses from the BMC. The IPMI messages are entered using printable ASCII

characters. While a user can enable a 'line edit mode' and directly enter the codes for an IPMI message, the main purpose of this mode is to facilitate the development of scripts that work with available terminal emulation programs.

Terminal Mode also supports a small number of ASCII Text Commands that can be used for operations such as getting a high-level hardware health status for the system, and doing system reset and power on/off operations.

13.2.1 PPP/UDP Proxy Operation

The BMC can support a mode that allows local system software (e.g. BIOS) to send and receive UDP datagrams via the BMC connection to the remote console. This operation is supported using two special message buffers associated with the channel: the PPP UDP Proxy Transmit Buffer and the PPP UDP Proxy Receive Buffer.

When PPP/UDP Proxy Operation is supported (and enabled) the BMC will check the destination port address used in incoming UDP datagrams. After removing any data escaping and checking the FCS, the BMC will check the destination port address in the UDP packet. If the packet is not addressed to either the primary or secondary RMCP Port addresses, the BMC will place the contents the packet into the PPP UDP Proxy Receive Buffer (assuming the packet fits, and the buffer is already empty). Otherwise, the packet will be silently discarded.

When sending messages to the remote console, local software loads the PPP UDP Proxy Transmit Buffer with the contents for the UDP message and then directs the BMC to deliver that message as a UDP datagram from the given serial/modem channel. The BMC fills in remaining data for the UDP and IP Header according to data passed in the *Send PPP UDP Proxy Packet* command and from the LAN Configuration parameters and then transmits the packet.

PPP/UDP Proxy Operation is only specified for execution via the BMC system interface. This capability is OPTIONAL for serial/modem channels that support PPP mode.

13.2.2 Asynchronous Communication Parameters

The asynchronous communication parameters consist of elements such as bit rate, type of handshake, parity, and other settings related to the configuration of the BMC's serial controller. These setting are configured via the serial/modem configuration parameters.

The number of different sets of parameters for a given channel depends on which messaging and alerting features are implemented:

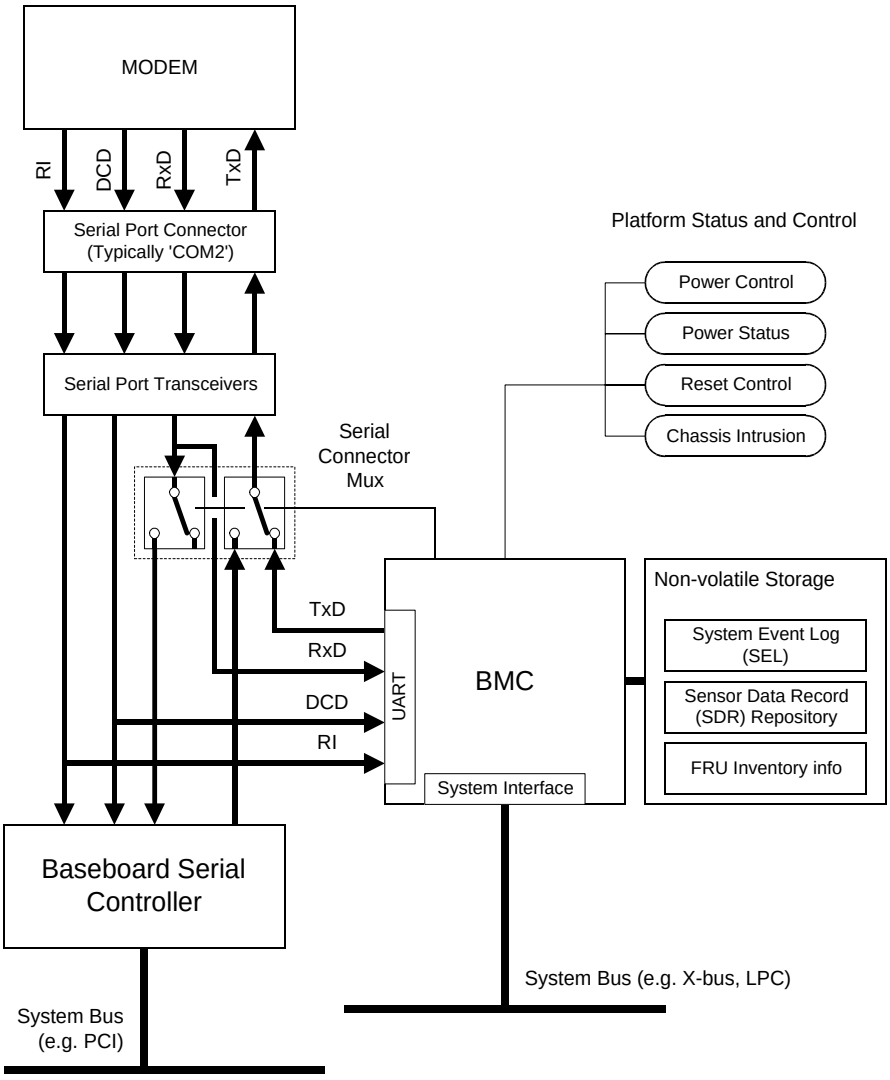
- There is one set for use by IPMI Messaging (Basic Mode, PPP Mode, or Terminal Mode) for the entire channel.
- There is one set of asynchronous communication parameters for each Alert or Callback Destination supported by a channel, used according to the Alert Type (Dial Page, TAP Page, PPP Alert, Callback).

13.2.3 Serial Port Sharing

Serial Port Sharing is an option where the BMCs serial controller and a baseboard serial controller can share the same serial connector. *Figure 13-1, Serial Port Sharing Logical Diagram*, presents a logical example of how this could be accomplished using a multiplexer to switch serial connector signals between the BMC and the baseboard serial controller.

Figure 13-1 is referred to as a logical diagram because this specification does not require a particular physical implementation as long as the commands function as described in this specification.

Figure 13-1, Serial Port Sharing Logical Diagram



13.2.4 Serial Port Switching

The following can cause a switch of the serial port:

Table 13-1, Serial Port Switching Triggers

From - To	Cause	disable[1]	Notes
BMC to Baseboard	IPMI *Set Serial/Modem Mux* command.	yes	
	Microsoft VT100 'Exit Exit UPS, ASIC or Service Processor' Escape sequence. (see ref: [MSVT]). Per [MSVT] the BMC should immediately acknowledge the switch to baseboard by returning <ESC>*	yes	<ESC>Q
Baseboard to BMC	IPMI *Set Serial/Modem Mux* command.	no	
	Microsoft VT100 'Invoke ASIC/Service Processor' Escape sequence (see ref: [MSVT])	yes	<ESC>(Pattern is also used for Connection Mode Auto-detect capability. See *13.2.9, Connection Mode Auto-detect*.
	Detection of basic mode *Get Channel Authentication Capabilities* request	yes	Requires Basic Mode to be enabled. Pattern is also used for Connection Mode Auto-detect capability. See *13.2.9, Connection Mode Auto-detect*.
	Detection of leading bytes in a PPP IPv4-UDP Packet addressed to BMC's IP address and RMCP primary or secondary port, ending at the RMCP message class field. PPP Protocol = Internet Protocol (e.g. 0021h) Packet = IPv4 UDP Datagram IP Address = BMC IP Address Port = Primary or Secondary RMCP port, as set in the serial/modem configuration parameters Initial packet data = RMCP v1.0 header with message class field = IPMI	yes	Requires PPP mode to be enabled. Pattern is also used for Connection Mode Auto-detect capability. See *13.2.9, Connection Mode Auto-detect*.
	RI Signal	yes	In Modem Connect mode only. See *13.2.11, Modem Activation* for more information.
	DCD Signal	yes	Used to cause a mux switch to BMC when in Direct Connect mode.

1. This indicates whether a configuration option to disable switching on this action exists. Note that switching may also be disabled as part of the operation of some access modes, such as 'Pre-boot Only'.

13.2.5 Access Modes

BMC channels used for serial/modem access can be configured for several Access Modes using the *Set Channel Access* command. The command determines which states of system operation (e.g. pre-boot) the channel can be used for BMC communication. Refer to section *6.6, Channel Access Modes* for more information.

13.2.6 Console Redirection with Serial Port Sharing

A common use for Serial Port Sharing is as a mechanism to allow the serial connection to be shared between the BMC and with BIOS Console Redirection. Serial Port sharing includes commands to help facilitate this application. This section presents an overview of those commands and how they might be used. This is just a starting point. The actual specification of the BMC with BIOS console redirection is outside the scope of this specification.

13.2.6.1 Detecting Who Answered The Phone

Since console redirection is normally used with a remote console, we'll start from when the remote console first connects to the system. Depending on the configuration, either the system or the BMC may be the party that 'answers the call'. Whether the BMC connects or not is determined by the access mode settings set via the *Set Channel Access* command, plus activation settings in the serial/modem configuration parameters. There is also a configuration parameter that allows the BMC to wait for a 'ring interval' before answering a call in order to give system software an opportunity to connect first.

Thus, when a remote console connects to a given system, it should first try to determine whether it connected with the BMC or with console redirection. This avoids the possibility that the remote console will send IPMI message data down only to have it be mis-interpreted as console-redirection keystrokes.

For Basic Mode and Terminal Mode, the BMC can be configured to periodically issue a *Serial/Modem Connection Active* message whenever it has the port and right after the port is switched to the BMC. The remote console can wait for this message as a confirmation that the BMC has the port before attempting to send any messages to the BMC. If that message is not received, it can assume that the system answered the phone instead of the BMC.

If PPP is used, PPP communication software on the remote console will typically initiate the PPP negotiation without waiting for the managed system. Because of this, there is less need to send a *Serial/Modem Connection Active* message first, since by the time the message is generated the remote console has already sent in characters. Similarly, because there are separate port addresses that would be used for RMCP traffic to the BMC, there's no strong need for the *Serial/Modem Connection Active* message to be periodically sent. Thus, the BMC does not send *Serial/Modem Connection Active* messages in PPP Mode except when the serial connection is being switched to or from the BMC.

When the serial connection is switched over to the BMC, the *Serial/Modem Connection Active* message will be delivered to the Primary RMCP port address and IP address of the remote peer that was negotiated during IPCP. If the BMC did not negotiate IPCP, the *Serial/Modem Connection Active* message will not be sent.

When the serial connection is being switched over to the system, the *Serial/Modem Connection Active* message will be delivered to the Primary RMCP port address and IP address of the remote peer that was negotiated with IPCP, and to each active session on that PPP channel. If the BMC did not negotiate IPCP, then the *Serial/Modem Connection Active* message will only be sent to the active sessions.

If remote console software wishes to detect the presence of a BMC, it can do so by sending a *Get Channel Authentication Capabilities* message after IPCP has been negotiated. [Note that if console redirection uses 'ASCII' then the remote console may have to assume that console redirection is occurring if it cannot establish a PPP Link. (Generally, ASCII text console redirection and PPP communication with the BMC don't share well together)]

If the system includes PPP Link authentication, the remote console could distinguish between whether the system or the BMC established the link based on the peer name that is used in the link negotiation.

13.2.6.2 Connecting to the BMC

The remote console can cause the connection to be switched back to the BMC by the mechanisms listed in *Table 13-1, Serial Port Switching Triggers*. Whether switching is allowed is based on what the Access Mode setting is for the channel. For example, if the channel is set to 'pre-boot only' - then the remote console will not be able to remotely switch the mux over to the BMC if the system is presently in run-time operation.

13.2.6.3 Connecting to the Console Redirection

The *Set Serial/Modem Mux* command provides the mechanism for a remote console to direct the BMC to switch the serial connection over to the system serial controller. The <ESC>Q sequence can also be enabled for this purpose.

13.2.6.4 Directing the Connection After Power Up / Reset

The remote console can send commands to the BMC to initiate a system power up or reset operation. After that operation, the remote console may want to see console redirection, or it may want to stay connected to the BMC. The *Set Serial/Modem Mux* command can be used to direct whether the remote console stays connected to the BMC or not. The *Set Serial/Modem Mux* command includes an option to allow a mux switch to be *requested* or to be *forced.* A mux switch that is requested may be denied (blocked). A BIOS using console redirection would typically *request* that the mux be switch over to the system during POST, so that the remote console could block that request if necessary. Thus, if the remote console wants to keep the connection, it simply issues a *Set Serial/Modem Mux* command to block requests to switch the mux to the system before sending the power up or reset command.

13.2.6.5 Interaction with Microsoft 'Headless' Operation

Microsoft has specified an interface [MSVT] for text-based console redirection to support pre-boot operations with the operating system. This specification includes escape sequences for activating and deactivating the connection to a 'service processor', as well as an escape sequence for hard resetting the system. See [MSVT] for more information.

IPMI v1.5 includes optional serial/modem configuration parameters for supporting [MSVT] in a system that implements serial port sharing along with [MSVT]. These parameters provide a common way for the [MSVT] activate/deactivate and reset sequences to be enabled or disabled in the system. Supporting these options in IPMI does not imply that a given implementation is conformant with the [MSVT] specification. Refer to [MSVT] for the full system requirements.

Note that the present [MSVT] specification calls out for a timeout on the escape sequence filtering. If an <ESC> is received, subsequent characters in the sequence must be received within 20 seconds.

13.2.6.6 Pre-boot Only Mode

The definition of Pre-boot Only access mode is that the BMC serial connection becomes disabled when the system starts to boot in order to guarantee that system software has full use of the serial connection without concern that incoming calls would be able to connect to the BMC. In order to provide emergency management coverage, someone using pre-boot only mode would typically also configure the watchdog timer and PEF so that a system power down or reset would occur on critical system failures, thus allowing a remote console to connect to the BMC.

The remote console has the ability to use the *Set Serial/Modem Mux* command to block mux switch requests, but allow mux switch 'forces'. This is typically used with the Pre-boot Only access mode. At the start of POST BIOS requests the mux. If a remote console is connected, it can block that request in order to continue to communicate with the BMC during POST, if desired, or the remote console can let BIOS take the mux in order to see BIOS console redirection.

At the conclusion of POST and start of boot, BIOS will typically force the mux away from the BMC and to the system. Once the mux has been forced away from the BMC in Pre-boot Only mode, the BMC is not allowed to take the port back until the next time the system is powered down or is reset.

Note that Alerting is *not* affected by Pre-boot Only mode. Alerting, if enabled, will 'take over the port' and cause an alert to be sent even if the system was using the port at the time. BIOS can use the *Get Channel*

Access command to determine that the BMC is configured to operate in Pre-boot Only mode for the serial connection.

13.2.6.7 Always Available Mode

In Always Available Mode the serial connection is considered to be dedicated to the BMC. In order to avoid confusion with run-time software, BIOS will typically hide or disable the serial port when the OS load process starts. BIOS can know when to do this by reading the access mode setting from the BMC using the *Get Channel Access* command.

13.2.6.8 Shared Mode

In Shared Mode the BMC is allowed to 'answer the phone' but run-time software is also able to use the serial connection when it's not being used by the BMC. BIOS can use the *Get Channel Access* command to see when the BMC is configured for Shared mode. In this case, it can leave the serial port enabled for run-time software access. The serial/modem configuration parameters include a 'ring interval' parameter that can be used to enable the BMC to only answer the phone if system software doesn't. This is accomplished by simply setting a ring interval for the BMC that is longer than the time it takes system software to answer.

13.2.7 Serial Port Sharing Access Characteristics

The following table lists the mux control and modem-answering characteristics according to the type of Access Mode and state of Serial Port Sharing. Corresponding mux switching steps that would typically be used in BIOS for supporting console redirection also listed.

In general, if a remote console application wishes to keep communication with the BMC after a power-up or reset, it should issue a *Set Serial Modem/Mux* command to block mux requests before issuing a *Chassis Control* command to cause the system to power-up or reset. However, it should not block mux 'force' operations because this could interfere with run-time access of the serial connection.

Table 13-2, Serial Port Sharing Access Characteristics

Serial Port Sharing	Access Mode	Characteristics
disabled	disabled	Same behavior for both Modem and Direct Connect Mode • Mux always set to system. • Set Serial/Modem Mux command is rejected. (See response data for the Set Serial/Modem Mux command). • Escape sequence / pattern triggered switching is not available. • Alerting Unavailable. • BMC Power-on Default = mux set to system BIOS Action at POST start: none required BIOS Action at POST end: none required
disabled	any except 'disabled'	Same behavior for both Modem and Direct Connect Mode • Mux always set to BMC. • Set Serial/Modem Mux command is rejected. (See response data for the Set Serial/Modem Mux command). • Escape sequence / pattern triggered switching is not available. • Alerting available. • BMC Power-on Default = mux set to BMC. BIOS Action at POST start: none required. BIOS Action at POST end: Recommend hiding/disabling baseboard serial controller.

enabled	disabled	Same behavior for both Modem and Direct Connect Mode • Mux always set to system (except during alerting). • Escape sequence / pattern triggered switching is disabled. • Set Serial/Modem Mux command available. • Alerting available. • BMC Power-on Default = mux set to system. BIOS Action at POST start: none required. BIOS Action at POST end: none required.
enabled	pre-boot	BMC pays attention to Modem Ring Time parameter until mux is *forced* to system using *Set Serial/Modem Mux* command. Afterwards, BMC will not automatically take over mux for IPMI messaging (will not answer the phone) until next power down or system reset (unless commanded). • Escape sequence / pattern triggered switching is available. • *Set Serial/Modem Mux* command available. • Alerting available. BMC will terminate call and automatically take the mux in order to send an alert, unless an IPMI Messaging Session is already in progress on the channel - in which case alert will be "deferred" until channel becomes available for dial-out. • BMC Power-on Default = Mux set to system if Modem Ring Time >00h and <3Fh, otherwise set to BMC. For Modem Mode, the BMC automatically takes over the connection upon power down, after system resets, and on detecting Ring based on Modem Ring Time parameter, except if a session is active - in which case the BMC will keep the connection (until the mux is *forced* to system using the *Set Serial/Modem Mux* command). - If Modem Ring Time parameter is >00h and <3Fh, mux will be set to system. BMC will take over mux if Ring Time expires while Ring is being detected via the RI signal. The mux will be returned to system when loss of DCD is detected, or if the BMC takes the mux but is unable to establish a connection. - If Ring Time = 00h, BMC will automatically take mux during power down and after system resets. BMC will also take mux and answer phone immediately when a Ring is detected via the RI signal. Mux will be claimed by BMC whenever loss of phone connection is detected. To the BMC, this is essentially the same 'phone answer' and power down/reset behavior as in 'Always Available' mode. For Direct Connect Mode the BMC automatically takes the connection upon power down, after system resets, and whenever loss of DCD is detected (if DCD-based switching is enabled) except if a session is active - in which case the BMC will keep the connection (until the mux is *forced* to system using the *Set Serial/Modem Mux* command). BIOS Action at POST start: Request mux to system if BIOS console redirection enabled. BIOS Action at POST end: Force to system. Keep baseboard serial controller enabled.
enabled	always available	• Escape sequence / pattern triggered switching is available. • *Set Serial/Modem Mux* command available. • Alerting available. BMC will terminate call and automatically take the mux in order to send an alert, unless an IPMI Messaging Session is already in progress on the channel in which case alert will be "deferred" until channel becomes available for dial-out. • BMC Power-on Default = Mux set to BMC. For Modem Mode, the BMC automatically takes over mux on power down, system resets, when loss of DCD is detected, and upon detecting initial activity of RI. The BMC also initializes the modem whenever DCD loss is detected. The BMC ignores the Modem Ring Time parameter. For Direct Connect Mode, the BMC automatically takes the mux upon power down, after system resets, and whenever DCD is absent (if DCD-based switching is enabled). BIOS Action at POST start: Request mux to system if BIOS console redirection enabled. BIOS Action at POST end: Force mux to BMC. Recommend BIOS hides/disables baseboard serial controller.

enabled	shared	• Escape sequence / pattern triggered switching is available. • *Set Serial/Modem Mux* command available. • Alerting available. BMC will terminate call and take mux in order to send an alert, unless an IPMI Messaging Session is already in progress on the channel - in which case alert will be "deferred" until channel becomes available for dial-out. • BMC Power-on Default = Mux set to system if Modem Ring Time >00h and <3Fh, otherwise set to BMC. For Modem Mode, the BMC controls mux upon power down, after system resets, and on detecting Ring based on Modem Ring Time parameter, except if a session is active - in which case the BMC will keep the connection: - If Modem Ring Time parameter is >00h, <3Fh. Mux will be set to system. BMC will take over mux if Ring Time expires while Ring is being detected via the RI signal. The mux will be returned to system when loss of connection (loss of DCD) is detected, or if the BMC takes the mux but is unable to establish a connection. - If Ring Time = 00h, BMC will take mux during power down and after system resets. BMC will also take mux and answer phone immediately when a Ring is detected via the RI signal. Mux will be claimed by the BMC whenever loss of DCD connection is detected. To the BMC, this is essentially the same 'phone answer' and power down/reset behavior as in 'Always Available' mode. For Direct Connect Mode, the BMC takes the mux upon power down and after system resets, except if a session is active - in which case the BMC will keep the connection. Once power is up, the BMC will leave the mux in the state last commanded by software or an escape sequence and will not automatically take the mux unless DCD loss is detected (if DCD-based switching is enabled), or an alert needs to be sent. BIOS Action at POST start: Request mux to system if BIOS console redirection enabled. BIOS Action at POST end: None. BIOS leaves mux setting alone. Note that the Boot Options contain flags that remote software can use to request BIOS to place the mux into a given setting at POST end.

13.2.8 Serial Port Sharing Hardware Implementation Notes

There are a number of characteristics that should be considered when designing hardware that aids in the implementation of serial/modem remote access and Serial Port Sharing

- The BMC needs the ability to monitor the DCD and RI signals from the serial connector in order to detect incoming modem calls.
- The physical implementation is required to ensure that the baseboard serial controller does not receive characters when the serial connector is switched to the BMC. The lines in to the baseboard serial controller should also be placed in an appropriate 'idle' level.
- In order to prevent signal transitions from causing interrupts to the baseboard communication routines, it is recommended that the remaining serial input signals to the baseboard serial controller also be switched and placed into an appropriate 'idle' level when the BMC is using the connector.
- The physical implementation is required to handle additional serial signal lines, such as RTS and DTR, in order to ensure that those signals remain in the active state to keep the modem connection active when switching between the baseboard and the BMC. If the BMC does not have control over those signal levels, it may be necessary to accomplish this using additional baseboard circuitry.
- The implementation is required to ensure that the serial connection does not see glitches or signal drops on the RTS, CTS, DSR, DTR, DCD, and RI lines due to switching between the baseboard and BMC serial controllers.
- It is up to the implementation to determine how it handles any 'Wake On Ring' options for the serial connector.

- The BMC may have other RS-232 lines under its control (DTR, RTS, CTS, and DSR). Hardware handshake via RTS and CTS is an implementation option. A BMC implementation may also optionally use DTR as an additional hang-up mechanism.

- The serial/modem feature is more valuable if the BMC can be communicated with when the system is in a powered-down or sleep state. This may require the port transceivers to be powered via Standby Power.

13.2.9 Connection Mode Auto-detect

Connection Mode Auto-detect refers the capability for the BMC to automatically detect and enter a particular Connection Mode (Basic mode, PPP mode, or Terminal mode) based on detecting an appropriate data pattern in the serial traffic from the remote console. Implementing Connection Mode Auto-detect is optional.

The configuration of this capability is handled via the Connection Mode parameter in the serial/modem configuration parameters. The parameters allow this capability to be disabled, and the BMC set to use just one connection mode for direct IPMI messaging to the BMC.

The following is the description and specification of the operation of Connection Mode Auto-detect.

- The BMC will auto-detect whenever an IPMI messaging session is not active. When a session is not active, the BMC constantly snoops for the different data patterns that will tell the BMC what connection mode to use, but also will cause the serial connection to be switched over to the BMC. The pattern matching routine must check for all supported mode patterns in parallel. For example, suppose the BMC supports auto-detect for all three connection modes. Even if it detects what looks like the start of the PPP Mode pattern, the pattern detection routine must continue to look for Basic Mode and Terminal Mode patterns in parallel until the PPP mode pattern is confirmed.

- For modem mode, the BMC detects that a connection has been established by detecting DCD or by receiving a ‘CONNECT’ string from the modem, depending on the implementation. For direct connect mode, the BMC uses DCD if DCD is enabled, otherwise it assumes that a connection exists any time the mux is switched to the BMC.

- If Basic Mode is enabled, and the mux is set to the BMC, the BMC will assume that Basic Mode is the desired connection type. The BMC will send out a *Serial/Modem Connection Active* “Ping” messages (if enabled) after the connection is established. A remote console that wishes to use Basic Mode should wait for the Ping before sending any packets to the BMC. This will avoid the possibility of the packet from being interpreted as keystroke input if the remote console happens to connect to text-based console redirection instead of the BMC.

- If the mux is already switched to the BMC, the BMC can detect a PPP Link Negotiation request and use that to set the connection mode to PPP Mode.

- The process is more complicated if system software performed the negotiation and the BMC is ‘snooping’ to see if it should be activated and enter PPP mode. The BMC needs to snoop for both compressed and uncompressed versions of the address and protocol fields. Note that PPP already specifies that a receiver must accept uncompressed headers even if compressed headers were negotiated, so this support should already be part of the BMC’s PPP routines. The BMC also needs to snoop according to the escaping that the system software negotiated. This is more problematic. The BMC needs to know what the system negotiated for its transmit ACCM (Asynchronous Control Character Mask) in order to know what control characters to ignore in the data stream. The following are options for handling this situation:

    a. Pre-configure the BMC to match the escape negotiation that software will use. The serial/modem configuration parameters contain a ‘Snoop ACCM’ parameter that the BMC can be directed to use. The Snoop ACCM indicates which control characters the BMC should ignore when snooping in PPP mode.

    b. Have the BMC snoop the Link Negotiation process. The BMC monitors the transmit ACCM that the system is using.

The following table lists the patterns that the BMC will look for when auto-detecting the connection mode. The patterns will also trigger a switch of the serial connector to the BMC if Serial Port Sharing is enabled.

Table 13-3, Auto-Connection Mode Patterns

Connection Mode	Pattern
Basic Mode	BMC looks for a complete *Get Channel Authentication Capabilities* command in basic mode format. The BMC should also respond to this command. Once a basic session is established the BMC will stay in basic mode until the session is terminated.
PPP Mode	If the mux is already connected to the BMC, the BMC will enter PPP mode if it detects the start of a PPP LCP packet after the connection is established. The BMC shall check the PPP packet bytes up to and including the LCP Packet Code field. An implementation can elect to ignore the Identifier and Data field values, but the Length and FCS (Frame check sequence) must be correct. If the mux is connected to the system, the BMC will switch and attempt to use PPP mode if it detects a PPP packet with the following characteristics: Protocol = IPCP Packet = IPv4 UDP Datagram IP Address = BMC IP Address Port = Primary or Secondary RMCP port, as set in the serial/modem configuration parameters Initial packet data = RMCP v1.0 header with message class field = IPMI An implementation can elect to either switch immediately on detecting this pattern without additional data integrity checks, or wait until it has verified the checksums and FCS on the packet. The BMC is not required to respond to the IPMI message that was encapsulated in the packet.
Terminal Mode	Terminal Mode can be enabled to be entered on receiving an “<ESC>(“ sequence (if enabled). The BMC will respond with [TMODE OK] and will operate in terminal mode until the connection is terminated or the data pattern for Basic Mode or PPP Mode IPMI-RMCP packet is detected.

13.2.10 Modem-specific Options

The serial/modem configuration parameters (set using the *Set Serial/Modem Configuration* command) support various modem configuration strings. These strings are set into non-volatile storage managed by the BMC. The BMC uses the strings to configure the modem for out-of-band access use.

Due to the limited length of these strings, it may not be possible to configure all necessary modem parameters. Rather than relying solely on these strings, it is recommended that the user pre-configure the modem for out-of-band operation and save those settings in the modem as the default. The Modem Strings can then be used just to hold strings that trigger the modem to restore its defaults.

The BMC automatically sends an <Enter> character (carriage return = 0Dh) after sending the *Modem Initialization* and *Hang Up Line* strings. <Enter> is not sent after the *Escape Sequence* string.

Table 13-4, Modem String Summary

String Name	Default String	Minimum. String Length	String Usage
Modem Init String	Implementation dependent. The string ATE1Q0V1X4&D0S0=0 is a good starting point.	64 bytes, including termination character	Transmitted every time the serial/modem connection becomes activated. The BMC automatically sends an <Enter> character after this string.
Modem Hang Up Sequence (unused if DTR hang-up is available and selected)	ATH	8 characters, plus termination character	Sent to modem whenever the BMC wants to terminate the session (i.e. password retry count is exceeded, etc). The BMC automatically sends an <Enter> character after this string. NOTE: If the DTR hang-up option is selected, this field will not be used.
Modem Escape Sequence (Unused if DTR hang-up is available)	+++	4 characters, plus termination character	Informs modem that next stream of bytes should be interpreted as command bytes and not sent to the remote software. The escape sequence must be sent prior to sending any other command if the modem is currently connected with another modem. *Note: This may cause the modem to hang up unless it has been configured otherwise.* Escape sequence is preceded by a 2 second pause in transmission from the BMC, and is follow by a 2 second pause in transmission. This sequence precedes the Modem Initialization string, except after a RI or connection after DCD loss.

13.2.11 Modem Activation

The BMC will monitor RI and claim the serial connection (switch the mux to the BMC) according to the Modem Ring Time parameter in the serial/modem configuration parameters[1] and *Section 13.2.7, Serial Port Sharing Access Characteristics*. After getting the connection, if DCD is already asserted, the BMC will monitor the incoming data

[1] A ring duration is used instead of a ring count in order to simplify handling the variations in RI that occur between different national telephone systems and modems.

stream for the start of an IPMI session. If DCD is not already asserted, the implementation can use one of the following mechanisms:

1. The BMC will initialize the modem with the initialization string from the serial/modem configuration parameters. The initialization string must be configured to set the modem to answer the phone. The BMC then waits for DCD to become active.

2. The BMC initializes the modem by sending the initialization string. The BMC then listens for a "RING" result code from the modem and sends out an "ATA" to answer the phone.

Of these two methods, method #2 is the preferred implementation since it does require leaving the modem in an auto-answer state.

13.3 Serial/Modem Connection Active (Ping) Message

If terminal-based console redirection is used, it is important for a remote console application to know whether the system or the BMC is connected to the serial connector before it sends any messages. Otherwise, if the serial port was already connected to the system, an incoming IPMI message could be interpreted by the system as redirected key strokes.

Therefore, there is a configuration option for Basic Mode and Terminal Mode that can direct the BMC to send out a *Serial/Modem Connection Active* request message **once every two seconds** whenever the serial connection is switched to the BMC, *with the first message starting immediately after the connection has been switched*.

This message is also referred to as the "Serial/Modem Ping". The message is used both to get the attention of the remote software and to allow the remote software to determine whether it is connected to the BMC or not. The Serial/Modem Connection Active message is primarily required when system console redirection is using a terminal-based format for input from the remote console, where incoming IPMI message characters could be misinterpreted as redirected input. For example, if console redirection was operating using a 'VT100' terminal emulation, the characters in an incoming IPMI message might be interpreted as a command or terminal control escape sequence.

If PPP is used, PPP communication software on the remote console will typically initiate the PPP negotiation without waiting for the managed system. Because of this, there is less need to send a *Serial/Modem Connection Active* message first, since by the time the message is generated the remote console has already sent in characters in an attempt to do link negotiation for PPP. Similarly, because there are separate port addresses that would be used for RMCP traffic to the BMC in PPP mode, there's no strong need for the *Serial/Modem Connection Active* message to be periodically sent. Thus, the BMC does not send *Serial/Modem Connection Active* messages in PPP Mode except when the serial connection is being switched to or from the BMC.

When the serial connection is switched over to the BMC, the *Serial/Modem Connection Active* message will be delivered to the Primary RMCP port address and IP address of the remote peer that was negotiated during IPCP. If the BMC did not establish the PPP Link, the *Serial/Modem Connection Active* message will not be sent.

When the serial connection is being switched over to the system, the *Serial/Modem Connection Active* message will be delivered to the Primary RMCP port address and IP address of the remote peer that was negotiated with IPCP, and to each active session on that PPP channel. If the BMC did not establish the PPP Link, then the *Serial/Modem Connection Active* message will only be sent to the active sessions.

> *Note:* If the BMC configured for any mode other than Direct Connect Mode, the *Serial/Modem Connection Active* message will not be sent out unless DCD is asserted. Sending *Serial/Modem Connection Active* messages while the modem is on-hook has been shown to prevent some modems from answering. This also implies that the modem should not be configured to hold DCD asserted.

13.3.1 Serial/Modem Connection Active Message Parameters

The *Serial/Modem Connection Active* message includes a parameter that indicates that a mux switch from BMC to the system serial controller is about to occur. This is provided to give a remote application some notification that the system is switching the port back over to the baseboard serial port. The *Serial/Modem Connection Active Message* with the 'switching to system' parameter will be sent out before the mux is switched and before the response is returned for the *Set Serial/Modem Mux* command.

13.3.2 Mux Switch Coordination

It is possible that the remote application committed an IPMI message for delivery to the BMC at the time that the switchover to the system occurred. If the mux switch occurred immediately, this means that the message might be delivered to the system instead of the BMC. To protect against this occurrence, the BMC can be configured to look for the remote console to acknowledge the *Serial/Modem Connection Active* message before the mux switch occurs.

There is a configuration option that directs the BMC to retry sending the *Serial/Modem Connection Active* message up to **three times with 20 ms between retries** if it does not get an acknowledge from the remote console. The BMC will then switch the mux if it has not received an acknowledge-message from the remote console within **three seconds** of sending the last retry.

The remote console acknowledges the switch by sending a *Serial/Modem Connection Active* request message of its own back to the BMC. The reason for this approach is so the BMC will return a response to the message, allowing the remote console to receive positive confirmation of the acknowledge message or to retry the message if the response is not received.

Note that the remote console should not send messages if it has not received a *Serial/Modem Connection Active* message or other message from the BMC in the last two seconds. The three second delay provides margin to help ensure that the console will not transmit with a *Serial/Modem Connection Active* message right when BMC times out and the mux switch occurs.

13.3.3 Receive During Ping

The serial/modem interface operates in a 'full duplex' mode. Thus, the BMC must continue to receive message characters while it is transmitting a *Serial/Modem Connection Active* 'Ping' message, or any other IPMI message.

For Basic Mode operation, the BMC must continue to handshake each message that it receives. This means that the BMC may insert an 'ACK' character in the middle of a Ping message transmission.

13.3.4 Application Handling of the Serial/Modem Connection Active Message

A robust Remote Console Application should be prepared to handle serial/modem remote access connection becoming deactivated or activated at any time.

A cessation of *Serial/Modem Connection Active* messages indicates that the serial/modem remote access connection is no longer active, while the occurrence of *Serial/Modem Connection Active* messages indicates that the connection is active. Thus, if a Remote Console Application should always monitor the presence/absence of *Serial/Modem Connection Active* messages, whether the serial/modem connection is active or not.

If the application is connected to the BMC, and does not receive an *Serial/Modem Connection Active* message within 2 seconds of its last transaction with the BMC should assume that the serial/modem connection has become deactivated.

Conversely, if the application is communicating with the system (e.g. console redirection) and an *Serial/Modem Connection Active* message is received, the application should recognize that the serial/modem connection has become reactivated.

13.4 Basic Mode

Basic Mode eliminates much of the overhead associated with PPP/UDP mode. Instead of encapsulating IPMI messages within an RMCP message in a UDP datagram in a PPP frame, the IPMI messages are simply encoded and framed for serial transmission. The price of this efficiency is that the remote console application cannot take advantage of built-in support for PPP and UDP in the operating system, but will need to implement IPMI communications routines on top of the OS's generic support for asynchronous serial communications.

Since Session IDs are not part of the basic IPMI message, *only a single IPMI session is supported in Basic Mode*. The BMC can use whatever Session ID value it wants for the *Get Session Challenge* and *Activate Session* commands.

13.4.1 Basic Mode Packet Framing

Framing is done with special characters to delimit the start and end of a Basic Mode packet, and to indicate the sequence for an escaped data byte (see following section). Framing and data byte escaping are applied *after* the message fields have been formatted. These special characters are specified in the following table.

Table 13-5, Basic Mode Special Characters

Description	Value
Start Character	A0h
Stop Character	A5h
Packet Handshake Character	A6h
Data Escape Character	AAh

Basic Mode messages can be thought of as IPMB messages with the I^2C start and stop condition framing replaced with start and stop characters, and with the addition of data byte escaping to ensure that the framing characters are not encountered within the body of the packet. The *Packet Handshake* character is a special value that is used for implementing a level of software flow control with the remote application accessing the BMC. See *Section 13.4.5, Packet Handshake*.

13.4.2 Data Byte Escaping

The *Start*, *Stop*, and *Escape* characters are disallowed within the body of the message. This is done to ensure that the start and end of a message is unambiguously delimited. If a byte matching one of the special characters is encountered in the data to be transmitted, it is encoded into a corresponding two-character sequence for transmission. This encoding is summarized in the following table.

Table 13-6, BASIC MODE Data Byte Escape Encoding

Data Byte	Encoded Sequence
A0h	AAh (ESC), B0h
A5h	AAh (ESC), B5h
AAh	AAh (ESC), BAh
A6h	AAh (ESC), B6h
1Bh <ESC>	AAh (ESC), 3Bh

The first character of the sequence is always the *Escape* character. Only the special Basic Mode characters plus the ASCII Escape <ESC> character, 1Bh, are escaped. (The ASCII Escape <ESC> character, 1Bh, is escaped to

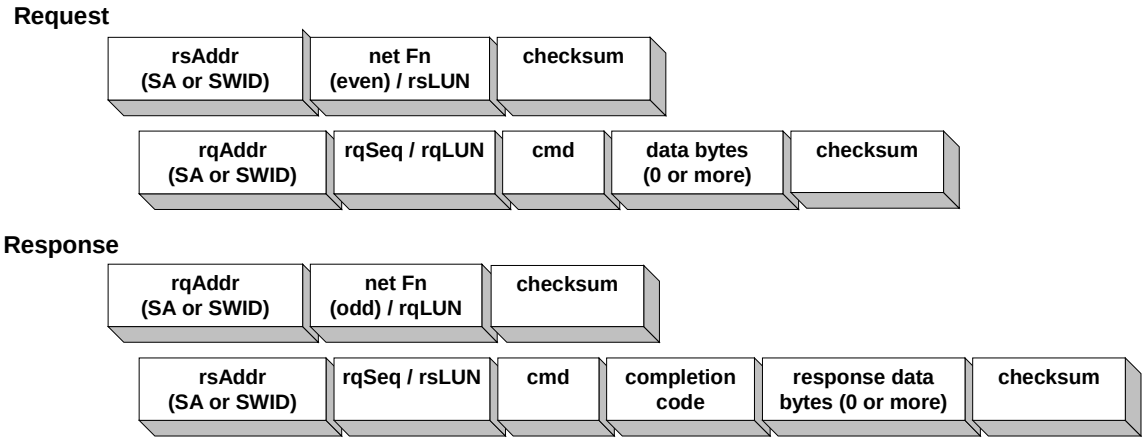
enable the BMC to snoop for certain escape sequences in the data stream, such as the <ESC>(and <ESC>Q patterns.) All other byte values in the message are transmitted without escaping.

When the packet is received, the process is reversed. If the two-byte 'escaped' sequence is detected in the packet, it is converted to the corresponding data byte value. The BMC shall reject any messages that have illegal character combinations or exceed message buffer length limits. The BMC may not send an error response for these conditions.

13.4.3 Message Fields

The message fields follow those used for the IPMB, as specified in the Intelligent Platform Management Bus Communications Protocol v1.0 Specification, with the exception of the Requester's (rq) Slave Address and Responder's (rs) Slave Address fields, which have slightly different definitions. Note, framing and data byte escaping are applied after the message fields have been formatted. The general message format is illustrated in the following figure:

Figure 13-2, Basic Mode Message Fields



Where:

checksum	2's complement checksum of preceding bytes in the connection header or between the previous checksum. 8-bit checksum algorithm: Initialize checksum to 0. For each byte, checksum = (checksum + byte) modulo 256. Then checksum = - checksum. When the checksum and the bytes are added together, modulo 256, the result should be 0.
cmd	Command Byte
completion code	Completion code returned in the response to indicated success/failure status of the request.
data	As required by the particular request or response for the command
LUN	The lower 2-bits of the netFn byte identify the logical unit number, which provides further sub-addressing within the target node.
netFn	Network Function code
rq	Abbreviation for 'Requester'.
rqLUN	Requester's LUN.
rqAddr	Requester's Address. 1 byte. LS bit is 0 for Slave Addresses and 1 for Software IDs. Upper 7-bits hold Slave Address or Software ID, respectively. This byte is 20h when the BMC is the requester.
rqSeq	Sequence number, generated by the requester.
rs	Abbreviation for 'Responder'.
rsLUN	Responder's LUN

rsAddr Responder's Slave Address. 1 byte. LS bit is 0 for Slave Addresses and 1 for Software IDs. Upper 7-bits hold Slave Address or Software ID, respectively. This byte is 20h when the BMC is the responder.

Seq Sequence number. This field is used to verify that a response is for a particular instance of a request. Refer to [IPMB] for additional information on use of the Seq field.

13.4.4 Message Retries

Basic Mode Messaging utilizes the same retry mechanisms used for the IPMB, as specification in the *Intelligent Platform Management Bus Communications Protocol v1.0 Specification*. The remote application timeout should be based on the IPMB timeout specifications, with additional time added for delay due to the phone system. A remote application can determine this additional delay for a given connection based on the time it takes to receive the *Handshake* character.

13.4.5 Packet Handshake

The handshake character is used to signal that the BMC has freed space in its input buffers for a new, incoming IPMI Message. The BMC typically returns a *Handshake* character within one millisecond of being able to accept a new message, unless the controller has already initiated a message transmission, or an operation such as firmware update has been initiated.

An implementation can either send the handshake character in the middle of the transmission or elect to wait to transmit the handshake character until the transmission in-progress has completed. If the implementation waits for the transmission to complete, the handshake character will typically be sent within one millisecond after the message transmission completed.

If the implementation elects to send the *Handshake* character in the middle of an outgoing message transmission, *it must not insert the Handshake character immediately following a Data Escape character*. The reason for this is to allow the remote console application some flexibility in whether it processes the *Handshake* character before or after removing data escaping.

13.5 PPP/UDP Mode

This mode of operation uses PPP [RFC1661] (point-to-point protocol) messaging for transmitting IP packets on an asynchronous link per [RFC1662] The following sections provide overview material on PPP operation in and explicit requirements for an IPMI implementation. The overview material is to provide context and a starting point for understanding the implementation.

The material does not supercede the PPP specifications. Designers are required to refer to the PPP RFC reference documents for information on implementing PPP, especially regarding the states involved in opening and terminating a PPP link.

All values are delivered most-significant byte first unless otherwise specified.

PPP/UDP mode transfers IPMI Messages encapsulated in RMCP Packets. This enables RMCP ASF Messages as well as IPMI Messages to be delivered to the BMC. The RMCP Packets are carried within UDP datagrams using the same format as the IPMI LAN messages. The resultant UDP datagrams are transferred within PPP frames. Specifications on this message formatting are provided in the follow sections.

13.5.1 PPP/UDP Mode Sessions

The BMC is only required to support one session on the PPP/UDP interface. A BMC implementation may elect to support multiple sessions.

13.5.2 PPP Frame Format

PPP/UDP mode framing follows [RFC1662]. [RFC1662] specifies an 'HDLC-like' format for PPP frames using on an asynchronous serial communication media. The following figure presents an overview of this format.

Figure 13-3, PPP Frame Format

Flag (7Eh)	Address (FFh)	Control (03h)	Protocol 1 or 2 bytes	Information	Padding	FCS 2 or 4 bytes	Flag (7Eh)	Inter-frame Fill or next Address

13.5.3 PPP Frame Implementation Requirements

Since the flag (7Eh) indicates both the start and end of a packet, it's possible that another flag could immediately follow a flag. However, the protocol also allows the 'end' flag to serve as both the end of one packet and the beginning of the next. The BMC must be able to handle both occurrences.

In order to reduce differences between implementations, it is recommended that the BMC must explicitly transmit a flag on both ends of the packet[2].

Support for the 16-bit (2-byte) FCS (frame check sequence) is mandatory.

[2] Per [RFC1662], only one Flag is required between frames, but if a two flag sequence is received, it is viewed as if an empty frame were received between two frames where the empty frame is silently discarded. Since the flag character delimits both the start and end of the frame, this requirement eliminates the need for the BMC to track that it had already sent a flag on the end of the previous frame and thus can skip sending a flag to start the current frame.

13.5.4 Link Control Protocol (LCP) packets

The following table presents a summary of the LCP Fields used in PPP. This is provided for reference only.

Table 13-7, LCP Code Fields

Code	LCP Packet Type
1	Configure-Request
2	Configure-Ack
3	Configure-Nak
4	Configure-Reject
5	Terminate-Request
6	Terminate-Ack
7	Code-Reject
8	Protocol-Reject
9	Echo-Request
10	Echo-Reply
11	Discard-Request

13.5.5 Configuration Requests

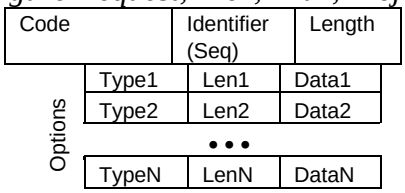
The first step in opening a PPP link is to establish the connection through the exchange of Configure packets. The PPP *Configure-Request* message is used to request changes to the link defaults. A *Configure-Request* is responded to with a *Configure-Ack, Configure-Nak,* or *Configure-Reject* packet. A link is considered established once the negotiation of link options has completed. If a Configure packet is received later, the link will be returned to the link establishment phase.

Table 13-8, Overview of PPP Configure-Ack, -Nak, & -Reject Packet Use

LCP Packet	
Configure-Ack	All options are recognized and accepted. The Options field is a copy of the Options field from the corresponding *Configure-Request*. Receipt of a Configure-Ack from both ends of the link signals that the link is opened and other (non-LCP) protocols may be accepted.
Configure-Nak	All options are recognized, but one or more are not acceptable. The Options field returns a list of the unacceptable options in the same order that the options were given in the *Configure-Request*. An implementation may append other options to prompt the peer to include those options in its next *Configure-Request* packet.
Configure-Reject	Some of the configuration options are not recognizable, or are not acceptable for negotiation.
…	…

The format of the *Configure-Request, -Ack, -Nak,* and *-Reject* packets follow the same format, as illustrated in *Figure 13-4, Configure-Request, -Ack, -Nak, -Reject Packet Format*.

Figure 13-4, Configure-Request, -Ack, -Nak, -Reject Packet Format



The **Code** field identifies whether the Link Control Packet is a *Configure-Request, -Ack, -Nak,* or *-Reject* packet, per *Table 13-7, LCP Code Fields.*

The **Identifier** field is similar to the IPMI 'Seq' field. The value must be changed for new requests, and the value in the request must be returned in the corresponding *Configure-Ack, Configure-Nak,* or *Configure-Reject* response.

The **Options** field holds a list of 0 or more link configuration parameters to be changed, and the corresponding values for those parameters. The Options field should only be filled with requests for non-default values. Options are not required to be in any given order in a *Configure-Request.* But the *Configure-Ack, -Nak,* and *-Reject* packets do need to return the options in the same order they were given in the corresponding *Configure Request.*

All Configuration Options that a sender wishes to negotiate are negotiated simultaneously.

Table 13-9, PPP Link Configuration Option Support Requirements

ID	Type	Len	Data	BMC Support Requirement
1	Maximum Receive Unit	4	bytes 1:2 - 2-byte value indicating request	Request: Recommended. It is recommended that the BMC request that packets smaller than 1500 bytes be used. The requested value must be ≥ XX bytes. IPMI only requires XX bytes, but OEM value-added features may require a larger maximum packet size. Response: Optional. The BMC can respond to a request that larger or smaller packets be used. If implemented, the BMC must reject/nak any request to use a value smaller than XX bytes. The implementation can elect to accept >1500 bytes, at the discretion of the implementer. Note that per PPP an implementation must accept at least 1500 bytes. See *Section 13.5.6, Maximum Receive Unit* Handling for more information.
2	Asynch Control Character Map			Request and Response: Optional
3	Authentication Protocol	≥4	bytes 1:2 - protocol ID C023 = PAP C223 = CHAP [RFC 1994] (Algorithm: #5 = CHAP w/MD5 [RFC 1994] #128 = MS-CHAP v1 #129 = MS-CHAP v2 bytes 3:N - data according to protocol ID	Request and Response: based on firmware support for PAP, CHAP, and MS-CHAP
4	Quality Protocol	≥4	bytes 1:2 - protocol ID C025 = Link Quality Report bytes 3:N - data according to protocol ID	Request and Response: Optional
5	Magic Number	6	bytes 1:4 - magic number	Request:Optional. Response: Recommended. It is recommended that the BMC indicate support for Magic Number.
7	Protocol Field Compression (PFC)	2	none	Request and Response: Recommended. It is highly recommended that the BMC support being configured to accept compressed protocol fields, and request that it can use compressed Protocol Fields when it transmits.
8	Address & Control Field Compression	2	none	Request and Response: Recommended. It is highly recommended that the BMC support being configured to receive compressed Address and Control Fields, and for the BMC to request that it can use compressed Address and Control Fields when it transmits.

1. PPP requires that implementations must be able to receive a full 1500-byte information field in case link synchronization is lost. If an MRU value is not specified, it is assumed to be 1500 bytes.

13.5.6 Maximum Receive Unit Handling

For full PPP compatibility, the BMC must accept up to 1500 bytes in the information field. The storage requirements could be even greater if the implementation elects to buffer the frame first and process escaped characters after getting the entire frame, instead of handling escaped characters as they're received.

Ideally, the BMC would have internal storage to hold this full amount of data, but this may not be economical in some implementations. If the BMC cannot directly buffer a 1500-byte MRU frame, the BMC must still continue to accept bytes until the end of the frame, and must not lose track of framing, terminate the link, or issue any error response just because of the overall frame length.

It is possible, though unlikely, that a UDP packet could contain an RMCP message that meets the BMC buffer size requirements, but is padded with additional bytes that cause the PPP Frame to exceed the BMC's buffer. An implementation can handle this possibility by continuing to calculate the FCS on characters received after the buffer has become full, before discarding those characters. Once the frame was completed, the BMC could check the leading contents of the buffer to see if a complete, valid message was contained in the initial bytes.

Note that there is a UDP Checksum that would also need to be tracked. However, the BMC could opt to ignore the UDP Checksum field. Barring formatting errors, the data-integrity-checking role of the UDP Checksum should be covered by the PPP FCS. The UDP Length field should also be able to be ignored for IPMI-RMCP messages, since the number of bytes preceding the IPMI Message Length field is constant, and the IPMI Message Length will indicate the number of valid bytes remaining in the message.

It is recommended that, for PPP frames containing RMCP/UDP packets, the implementation accept PPP frames greater than its buffer size, track and verify the Frame Check Sequence, and attempt to validate and interpret the leading, buffered data.

13.5.7 Protocol Field Compression Handling

The least significant bit of a protocol field indicates that the last byte of the protocol has been sent. Therefore, if the ls-bit of the first protocol byte position is a '1', the implementation can simply assume that the protocol field has been compressed to one byte.

Accepting a configuration request for Protocol Field Compression indicates that the implementation supports receiving compressed protocol field values. This does not obligate the transmitter to send them. Thus, the receiver must be able to receive frames that use both compressed and non-compressed formats.

It is recommended that the BMC request that it can use Protocol Field Compression for the frames it sends. If this configuration option is accepted, the BMC itself should use it. Note there may be some cases where the BMC may need to transmit without using compressed fields, even though it has negotiated for compressed fields to be accepted. This is allowable in PPP and is also allowable in a BMC implementation.

13.5.8 Address & Control Field Compression Handling

Per the PPP specification, when Address & Control Field compression is used the Address and Control fields are simply omitted. On reception, the Address and Control fields are decompressed by examining the first two bytes. If they contain the values 0xff and 0x03, they are assumed to be the Address and Control fields. If not, it is assumed that the fields were compressed and were not transmitted.

This works because the first byte of a two byte Protocol field will never be 0xff (since it is not even), and the Protocol field value 0x00ff is not allowed (reserved) to avoid ambiguity when Protocol-Field-Compression is enabled and the first Information field byte is 0x03.

LCP Packets are not allowed to be sent with compressed Address and Control fields.

Accepting a configuration request for Address and Control Field Compression indicates that the implementation accepts frames using Address and Control Field Compression. This does not obligate the transmitter to send them. Thus, the receiver must be able to receive frames the use both compressed and non-compressed formats.

It is recommended that the BMC request that it can use Address and Control Field Compression for the frames it sends. If this configuration option is accepted, the BMC itself should use it. Note there may be some cases where the BMC may need to transmit without using compressed fields, even though it has negotiated for compressed fields to be accepted. This is allowable in PPP and is also allowable in a BMC implementation.

13.5.9 IPMI/RMCP Message Format in PPP Frame

IPMI Messages are carried in RMCP Packets in UDP using the same format as the IPMI LAN messages. This enables RMCP ASF Messages as well as IPMI Messages to be delivered to the BMC. RMCP support adds only four bytes overhead to an IPMI Session message in UDP.

Figure 13-5, IPMI Message in PPP Frame Format

	Field	Size	Value
PPP Frame	Flag	1	7Eh
	Address	1 or 0[1]	FFh[1]
	Control	1 or 0[1]	03h[1]
	Protocol	1 or 2[2]	0021 = IPv4
IP Header	Version and Header Length	1	
	Service Type	1	
	Total Length	2	
	Identification	2	
	Flags & Fragment Offset	2	
	Time to Live	1	
	Protocol	1	11h
	Header Checksum	2	
	Source IP Address	4	
	Destination IP Address	4	
UDP Header	Source Port	2	
	Destination Port	2	26Fh
	UDP Length	2	
	UDP Checksum	2	
RMCP Header	Version	1	
	Reserved	1	
	RMCP Sequence Number	1	FFh for IPMI[3]
	Class of Message	1	07h for IPMI
IPMI Session	Authentication Type	1	
	Session Sequence #	4	note[5]
	Session ID	4	note[5]
	Message Authentication Code (AuthCode) Not present when Authentication Type = none.	16	
	IPMI Message Length	1	
IPMI Message	Per *Section 12.4, IPMI LAN Message Format*	varies	
PPP Frame	FCS	2	
	Flag	1[4]	

1. Dependent on whether Address & Control Field Compression is used
2. Dependent on whether Protocol Field Compression is used
3. RMCP Messages with class=IPMI should be sent with an RMCP Sequence Number of FFh to indicate that an RMCP ACK message should not be generated by the message receiver.
4. Per [RFC 1662] “Each frame begins and ends with a Flag sequence… Only one Flag Sequence is required between two frames. Two consecutive Flag sequences constitute an empty frame, which is silently discarded an not counted as an FCS error.” The implementation should take care to track that a single flag character may indicate both the end of the present packet, and the start of the next.

5. The Session ID and Session Sequence Number must be non-zero for commands executed during an active session. All 0's for the Session ID and/or Session Sequence Number (null Session ID, null Session Sequence Number) are special values only used for commands that can be executed prior to establishing a session, e.g. *Get System GUID*, *Get Channel Authentication Capabilities*, and *Get Session Challenge*. The *Activate Session* command uses a null Session Sequence Number before a session is activated, but does not use a null Session ID. Instead, it must use the Temporary Session ID given by the BMC in the response to the *Get Session Challenge* command.

13.5.10 Example of IPMI Frame with Field Compression

A PPP frame for IPMI in UDP/RMCP that uses both Protocol and Address-and-Control Field compression will have the following format. Note that per PPP, uncompressed frames must also be accepted at any time. Buffer sizes must take this into account.

Figure 13-6, IP Frame with Field Compression

Flag (7Eh)	Protocol (21h = IPv4)	UDP/RMCP/IPMI Packet data	FCS 2 bytes	Flag (7Eh)

13.5.11 Frame Data Encoding

In order for the Flag and Control-Escape bytes to be utilized, they must not appear directly in the data stream. The encoding of utilizes an escaping mechanism where bytes in packet are replaced with a two-character sequence in order to prevent them from being mis-interpreted as being flag or Control-Escape bytes. The escaping mechanism can also be used to prevent bytes from being interpreted as ASCII control characters.

Only bytes between flag bytes are escaped. The flag byte themselves are never escaped.

13.5.12 Escaping Algorithm

To 'escape' a character, N, the BMC simply emits a 7Dh character, followed by N exclusive-OR'd with 20h. To convert the escaped-pair back to the original data byte, the 7Dh is thrown away and the second character exclusive-OR'd with 20h.

13.5.13 Escaped Character Handling

By default, the following characters are escaped:

Table 13-10, Default Escaped Characters

Character	value	Escaped as:
Control Escape	7Dh	7Dh, 7Dh
Flag	7Eh	7Dh, 5Eh
ASCII Control Characters	00h-20h	7Dh, (value XOR 20h)

The BMC must ignore non-escaped versions of the above characters as part of the frame data, unless there has been a negotiation that allows some of the characters to be sent without escaping. (See *Section 13.5.14, Asynch Control Character Maps (ACCM),* below) The control-escape character and flag characters still need to be interpreted, of course.

The reason that non-escaped characters are dropped is that an intervening communication device may have inserted the characters.

Only non-escaped characters are eligible to be dropped on receipt as spurious characters in the frame data. The BMC must accept all escaped characters received within flag delimiters as part of the frame data.

13.5.14 Asynch Control Character Maps (ACCM)

PPP includes an option that allows the negotiation of which characters require escaping and which can be optionally escaped. This is accomplished by negotiating ACCMs (Asynch Control Character Maps) between both ends of the link. ACCM negotiation can potentially improve data throughput by reducing the number of characters that require escaping.

The BMC is not required to support ACCM negotiation. If ACCM negotiation is not supported, the BMC must handle character escaping and escaped characters as described in *Section 13.5.13, Escaped Character Handling*.

ACCMs are negotiated using a 32-bit parameter where each bit corresponds to a character from 00h to 1Fh, with the least-significant bit corresponding to 00h. When sent in a configure-request, the ACCM indicates which values that the originator of the request must receive in escaped format, and which can optionally be sent without escaping. This ACCM is referred to as the 'receiving ACCM'.

The 'sending ACCM' is the set of characters that will be sent with escaping, barring any additional configuration due to a configure-request. By default, the sending ACCM is FFFFFFFFh - indicating the set of default escaped characters listed in *Table 13-10, Default Escaped Characters*. The sending ACCM can only change as the result of receiving a configure-request indicating that fewer characters need to be escaped. Note that a transmitter can send escaped codes for values >1Fh (in addition to the required escaping of the flag and control-escape values). These cannot be configured via negotiation, however.

The receiver of the configuration-request can respond with a configure-ack for the option, or it can responds with a configure-nak and return the union of the requested ACCM with the ACCM that it will be using for transmission. For example, suppose the remote console was hard-coded to always escape character 0Dh for some reason. If the BMC submitted an ACCM indicating that it required only characters 03h and 04h to be escaped, the remote console could respond with a configure-nak that it would always escape 03h, 04h, and 0Dh. This would tell the BMC that it should also ignore 0Dh characters in the data.

The receiving ACCM is assumed to be FFFFFFFFh by default. That is, a transmitter must escape values 00h-1Fh (plus flag and control-escape values encountered in the frame data) unless it receives a configure-request indicating that certain values do not need to be escaped. This also means that the receiver can expect to receive 00h-1Fh in escaped format until it has successfully configured an alternative.

13.5.15 IP Network Protocol Negotiation (IPCP)

Once the PPP link has been established, it is necessary to send NCP (network control packets) to choose and configure one or more network layer protocols.

[RFC1332] describes IP Control Protocol (IPCP). IPCP is a network control protocol used to choose transfer of IPv4 packets via PPP. The BMC must both accept IPCP configuration requests and generate IPCP configuration requests.

- BMC **shall** Configure-Ack an IPCP Configure-Request that contains 0 (zero) configuration options.
- BMC **shall** issue a Configure-Request for IPCP option 3 (IP Address) to request that the IP Address specified by the PPP IP Address parameter (see *Table 20-4, Serial/Modem Configuration Parameters*) be used as the BMC's IP Address.
  - If IP Address Assignment is enabled in the serial/modem configuration parameters, the BMC **shall** accept an address assignment that is returned via a corresponding Configure-Nak from the remote console.
  - If IP Address Assignment is not enabled (Fixed IP Address), the BMC **shall not** accept a different address assignment returned via a corresponding Configure-Nak from the remote console. If the BMC PPP IP Address is not accepted, the BMC shall issue a new Configure-Request with the same PPP IP Address value (but a new identifier value). This will be repeated until the PPP IP Address parameter is

accepted, or at least three Configure-Requests for setting the PPP IP Address parameter have been issued.

- The BMC **shall** silently discard later IP Protocol (0021h) UDP packets that are addressed to the Primary or Secondary RMCP ports, but do not match the negotiated PPP IP Address.
- BMC **shall** accept IPCP option 3 (IP Address) in an IPCP Configure-Request, unless that message matches the BMC's PPP IP Address.
- BMC **shall** Configure-Nak IPCP option 3 if IP Address Assignment is disabled, and return the PPP IP Address value from the serial/modem configuration parameters in the Configure-Nak.
- BMC **shall** Configure-Reject IPCP option 1, IP Addresses. This option has been deprecated in IPCP.
- BMC **shall** Terminate-ACK a Terminate-Request for IPCP.
- The BMC **shall not** transmit IPv4 protocol (0021h) packets after an IPCP Terminate-Request has been received, until IPCP is renegotiated.
- The BMC **shall silently discard** any IPv4 protocol packets received after an IPCP Terminate-Request has been received, until IPCP is renegotiated (the BMC receives its first IPCP Configure-Request).
- BMC **may** accept other IPCP options to support OEM features. For example, Option 2 is Van Jacobsen compression for TCP/IP. Remote stacks must be prepared for the potential that a BMC implementation might request network protocols and/or configuration options beyond those specified in this document.
- BMC **shall not** enable OEM framing extensions alongside PPP mode. It is possible that an OEM may want to include proprietary serial framing formats or special handshake or escape sequences that are not specified in this document, but that work as a proprietary extension to PPP mode. The BMC will not be considered to be conformant for PPP mode if these extensions are active. It is acceptable for the BMC to have an OEM-specific option to enable/disable OEM extensions. In this case, conformance will only be assessed when such OEM extensions are disabled. Remote stacks and remote console applications designed for IPMI may break when OEM extensions are enabled.
- The BMC **shall** Request the remote console IP Address by issuing a configure-request for Option 3 with an IP Address value of 00.00.00.00 [This provides a mechanism for the BMC to obtain the remote console connection's IP Address in order to enable the BMC to asynchronously send UDP datagrams to the remote console.]
- The BMC **shall** accept IPv4 Protocol Packets (0021h) once it has received and responded to an IPCP Configure-Request from the remote console.

13.5.16 CHAP Operation in PPP Mode

An implementation can support CHAP as a mechanism for authenticating the serial/modem connection at the link level. This option is separate from the whether or not RMCP/IPMI Message packets are authenticated once the link has been established. Serial/modem configuration options to select and support either standard CHAP [RFC 1994], MS-CHAP v1 [RFC 2433], or MS-CHAP v2 [RFC 2759] are provided.

There are two classes of configuration options for CHAP:

- IPMI Messaging: There is a single set of options that configures what type of CHAP, if any, is used for serial/modem IPMI messaging with the BMC in PPP Mode. These are referred to as PPP Link options in the serial/modem configuration parameters. See *Table 20-4, Serial/Modem Configuration Parameters*.
- Callback and Dial-out Alerting. Another class of PPP options relates to user names and accounts for connecting with a remote system via a PPP-to-LAN connection. The specification supports multiple sets of these options. The option sets are grouped under a *PPP Account Selector* number in the configuration

parameters. The PPP Account Selector provides the link that associates a set of PPP account parameters with a particular serial/modem Alert or Callback destination.

The *Set/Get User Name*, *Set/Get User Access*, and *Set/Get Channel Access* commands are used to configure the password and username associated with CHAP link-level authentication for IPMI messaging in PPP mode. Note that the same User Names and passwords that are used for link authentication can either be the same as those used for IPMI Messaging, or they can be different. There is a bit setting associated with the *Set User Access* command that determines whether the information associated with a given User ID is to be used for PPP Link Authentication (e.g. CHAP), or IPMI Messaging Authentication, or both.

PPP Account 1 is used to hold information for both IPMI Messaging via PPP and for callback, such as the IP Address that the BMC will attempt to negotiate for itself.

13.6 Serial/Modem Callback

Callback provides a serial/modem channel mechanism that enables a remote console to direct the BMC to call a pre-configured destination and attempt to establish an IPMI Messaging connection. Callback provides both a security enhancement and a way to 'reverse' phone charges associated with managing a system.

Callback is primarily for use under the Modem connection mode. It can, however, be used with Direct Connect mode for testing and development purposes. For example, PPP destination parameters could be tested locally without requiring going through a modem. It's potentially possible to locally verify parameters or do testing by looping back from one system serial port to another using a 'null modem' cable.

Once the callback connection has been established, the BMC waits for the remote application to activate a session with the BMC by issuing a *Get Session Challenge* command, etc.

If the *Serial/Modem Connection Active* (Ping) message is enabled, the BMC will announce its presence by periodically sending the Ping once the connection has been established. The call will be automatically terminated if the remote system does not activate a session with the BMC within the Session Inactivity Timeout interval for the channel (See *Section 6.11.13, Session Inactivity Timeouts*).

IPMI Messaging and Callback use the same mode setting (basic mode, PPP mode, or terminal mode). I.e. you can't request callback using one messaging mode, and have the BMC connect using a different messaging mode. The Callback function is implemented at the IPMI Message level. PPP Callback (I.e. PPP LCP option 0D) is not used. For callback, the PPP Account Set settings parameters are only used if IPMI Messaging for the channel is set to PPP Mode.

In order to initiate a callback, the remote console first connects to the BMC using a pre-configured User ID and then issues the *Callback* command. The User ID can be restricted to 'Callback level' privilege so that the only operation that can be performed is to initiate a callback using the *Callback* command. A User ID can also be restricted to only be accessible while a callback connection is active. Together, this provides the option to allow one User ID and password to initiate the callback, while making it necessary to have a callback connection active in order to perform any higher-privilege level connections to the BMC.

The *Set User Access* command is used to configure, on a per channel basis, whether a given user is enabled, what the user's limits are, and whether user access is restricted to only being available during a callback connection.

The Callback, Operator, and Administrator privilege levels can be used to initiate a Callback, but the User Level cannot. This is consistent with the definition of User privilege.

13.6.1 Callback Control Protocol (CBCP) Support

An implementation that supports PPP can elect to support the Microsoft Callback Control Protocol (CBCP). CBCP is a Microsoft Corporation specification for supporting callback from a Microsoft RAS (Remote Access

Services) PPP connection. Other devices such as serial-to-LAN gateways may also support CBCP. See [CBCP] for specification information.

With respect to the BMC, CBCP provides a protocol by which a remote console can request the BMC to initiate a callback to the remote console.

CBCP is negotiated during the initial LCP phase. Once CBCP has been negotiated, per [CBCP] the BMC initiates the callback process by issuing a request that tells the remote console what callback number options are available from the implementation.

A BMC implementation can support one or any combination of the callback number options listed in the following table. An implementation may elect to implement CBCP callback numbers such that different users can have different callback numbers, or where callback numbers are shared across users.

Table 13-11, CBCP Callback Number Options

Option	Description
No Callback	The remote console requests not to be called back at all.
Callback to caller-specified number	The BMC indicates that it allows the remote console to specify which number is to be called back.
Callback to a pre-specified number	The BMC calls back a pre-configured phone number. If a PPP Link authentication protocol such as CHAP is used, the BMC uses the user id string from the authentication negotiation to look up which phone number to use for the given user. Otherwise, a global number associated with the serial/modem configuration parameters for the channel will be used.
Callback to one from a list of numbers	The BMC offers up a list of possible phone numbers that the callback can be directed to. The remote console picks one and returns it to the BMC. If the number matches one from the list, the BMC calls that number. If a PPP Link authentication protocol such as CHAP is used, the BMC uses the user id string from the authentication negotiation to look up which set of phone numbers to offer to the given user. Otherwise, a global set of numbers associated with the serial/modem configuration parameters for the channel will be used.

13.6.1.1 CBCP Address Type and Dial String Characters

CBCP includes an Address Type field that indicates the format used for callback addresses. Address Type = 1 indicates PSTN/ISDN. No other Address Type values are specified, therefore this field is, by default, a fixed field for IPMI implementations.

Per [CBCP] callback strings are null terminated ASCII strings formed from the following set of characters:

0-9, *, #, T, P, W, @, comma, space, dash, and parentheses.

This specification applies to using NT RAS as the dialer. For IPMI 1.5, however, the BMC is the dialer. Thus, additional characters specified in section *13.11.1, Alert Strings for Dial Paging* can also be used in the Dial String for CBCP callback.

IPMI 1.5 implementations do not check for illegal characters in dial strings. It is the responsibility of configuration software to ensure that correct characters are entered.

13.7 Terminal Mode

Terminal Mode is an operating mode of a serial/modem channel used for the following purposes:

- It provides a printable text-based mechanism for delivering IPMI between a terminal or remote console and the BMC. The text-based approach makes it simpler to develop script-based tools for generating and handling IPMI messages.
- It provides a small number of ASCII-text based commands to enable a small number of basic recovery and status functions to be executed when only a dumb terminal is available in lieu of real system management software.

13.7.1 Terminal Mode Versus Basic Mode Differences

Terminal Mode is primarily intended for local use rather than remote use via a modem. The following are the main differences between terminal mode and basic mode operation:

- If password protection is desired, only 'plain text' passwords can be used. Password characters are restricted to be from the printable set of ASCII characters as defined in *Appendix E - Terminal Mode Grammar.*
- Passwords can be entered two ways in Terminal Mode: either via the *Get Session Challenge / Activate Session* command, or via an ASCII Text command.
- Terminal Mode does not utilize checksums on IPMI messages or ASCII Text commands. If a modem connection is used, the modem should be configured for error correction, or Basic Mode should be used instead.
- Terminal Mode remote console is limited to a single, fixed, single Software ID (SWID). See *Table 5-4, System Software IDs.* The fixed SWID is used where a requester's SWID would have been extracted from the IPMI Message. For example, if Terminal Mode IPMI Messaging is used to generate a Platform Event Request message (Event Message) the SEL Record would contain the fixed SWID identifying the Terminal Mode remote console.
- The Terminal Mode remote console is limited to a single LUN (00b). This LUN is implicit in the message format. When Terminal Mode request or response messages are bridged to other media, the value 00b is used as requester's or responders LUN, respectively.
- Terminal Mode messages delivered to SMS via BMC LUN 10b always go to SMS Software ID 20h (41h) LUN 00b, unless the *Send Message* command is used to put the message in the receive message queue.
- Callback is not supported for Terminal Mode. You can trigger a callback from Terminal Mode, but the party that is called must support either Basic Mode or PPP Mode.

13.7.2 Terminal Mode Message Format

Terminal mode messages are of the general format:

[<message data>]<newline>

The left-bracket and right-bracket+<newline> characters serve as START and STOP delimiters for the message. Note that the right-bracket and <newline> characters together form the sequence that indicates the end of the message. <newline> characters may appear within the message as a result of input line editing and multi-line output message data.

13.7.3 IPMI Message Data

IPMI Messages are sent and received in Terminal Mode <message data> as a series of case-insensitive hex-ASCII pairs, where each is optionally separated from the preceding pair by a single <space> character. The following is an example of an IPMI Request message in Terminal Mode:

[18 00 22]<newline>

The Terminal Mode Request Message field definitions follow those used for the Basic Mode except that there is no Slave address / Software ID field or LUN information for the requester. The software ID and LUN for the remote console are fixed and implied by the command. The SWID for messages to the remote console is always 40h, and the LUN is 00b.

Instead, there is a 'bridge' field that is used to identify whether the message should be routed to the BMC's bridged message tracking functionality or not.

Figure 13-7, Terminal Mode Request to BMC

Byte 1	Byte 2	Byte 3	Byte 4:N
NetFn (even) / rsLUN=00b (BMC)	rqSeq / Bridge=00b (BMC)	cmd	data

The following figure shows the corresponding format of a response message from the BMC.

Figure 13-8, Terminal Mode Response from BMC

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5:N
NetFn (odd) / rsLUN=00b (BMC)	rqSeq/Bridge=00b (BMC)	Cmd	Completion Code	Data

13.7.4 Terminal Mode IPMI Message Bridging

The terminal mode message includes a 'bridge' field that is used to determine whether the message is going to or coming from the BMC's command functionality, or to/from the BMC's 'bridge' tracking functionality.

The message is interpreted based on the value of the bridge field, whether the message is a request or response, and the message direction per the following table.

> Note that messages to and from the system interface are transferred using the BMC SMS LUN, 10b, with the bridge field set to 00b.

Support for Terminal Mode IPMI Message Bridging is optional.

Table 13-12, Terminal Mode Message Bridge Field

Bridge Field	Request/ Response	Message Direction (to BMC)	LUN	Message Interpretation
00b	Request	In	00b, 01b, 11b	**Remote Console request to BMC functionality** Message is a request from the remote console to the BMC
00b	Response	Out	00b, 01b, 11b	**Response to Remote Console from BMC functionality** Message is a response to an earlier request from the remote console to the BMC
00b	Request	In	10b	**Remote Console request to SMS** Message is a request from the remote console to SMS via the Receive Message Queue
00b	Response	Out	10b	**SMS Response to Remote Console** Message is a response to an earlier request from SMS
00b	Request	Out	00b, 01b, 11b	**Asynchronous Request to Remote Console from BMC**
00b	Response	In	00b, 01b, 11b	**Remote Console Response to earlier Asynchronous Request from BMC**
00b	Request	Out	10b	**Asynchronous Request from SMS to Remote Console**
00b	Response	In	10b	**Remote Console Response to earlier Asynchronous Request from SMS**
01b	Request	In	any	**ILLEGAL COMBINATION** The remote console bridges requests to other media by encapsulating the message content in a *Send Message* command to the BMC functionality.
01b	Response	Out	any	**Response to earlier Bridged Request from Remote Console** Message is the asynchronous response from an earlier bridged request that was encapsulated in a *Send Message* command issued to the BMC by the remote console.
01b	Request	Out	any	**Asynchronous, bridged request to remote console from other media** Message is a bridged request to the remote console from another media, e.g. the system interface. BMC assigns the sequence number as part of bridging.
01b	Response	In	any	**Remote Console response to earlier asynchronous request from another media** Message is a response from the remote console to an earlier bridged request from another media. BMC uses the sequence number in the response to determine how to route the response to the original requester.

13.7.5 Sending Messages to SMS

Terminal Mode uses BMC LUN 10b to send messages to SMS (system interface) via the Receive Message Queue in the BMC. The following shows the format of a request message delivered to SMS, and the corresponding response.

Figure 13-9, Terminal Mode Request to SMS

Byte 1	Byte 2	Byte 3	Byte 4:N
NetFn (even) / rsLUN=10b (BMC to SMS)	rqSeq =XX / Bridge=00b (BMC)	cmd	data

Figure 13-10, Terminal Mode Response from SMS

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5:N
NetFn (odd) / rsLUN=10b (BMC to SMS)	rqSeq =XX / Bridge=00b (BMC)	cmd	completion code	data

13.7.6 Sending Messages to Other Media

The Send Message command is used to deliver a message to a different media, e.g. IPMB. The following figure illustrates the data contents that would be used in a Send Message command to deliver a Terminal Mode request to another (non-system interface) channel:

This data would be carried in a *Send Message* command of the following format:

Figure 13-11, Send Message Command for Bridged Request

NetFn (even) / BMC_LUN	rqSeq / bridge=00b	cmd = Send Message		
channel #	session handle	rsSWID		
netFn (even)/**rsLUN**	chk1	rqSWID=81h (terminal mode console)	**rqSeq/**rqLUN=00b	**cmd**
<data>	chk2			

The BMC will return a *Send Message* response matching the *Send Message* request. This will normally be returned immediately after the request.[3]

Figure 13-12, Response to Send Message Command for Bridged Request

NetFn (odd) / BMC_LUN	rqSeq / bridge=00b	cmd = Send Message	completion code = 00h (OK)

Later, the bridged response will be returned. The following figure shows the contents of a corresponding bridged response to the Remote Console:

Figure 13-13, Bridged Response to Remote Console

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5:N
netFn (odd) / **rsLUN**	**rqSeq** / bridge=01b	**Cmd**	Completion Code	Data

Note that much of the targets addressing information (rqSWID, rqLUN) is absent from the response. The remote console must use the original request's sequence number (rqSeq), netFn/rsLUN, and command values to match bridged response up with the earlier bridged request. These fields are highlighted with **bold** in the preceding *Send Message* and *Bridged Response* figures.

[3] Note that because IPMI messaging allows for other messages to appear between requests and responses, it is possible that one or more asynchronous messages could appear between the *Send Message* request and response. Console software should be prepared to handle such occurrences.

13.7.7 Terminal Mode Packet Handshake

There is a configuration option that allows the BMC to output a character sequence that indicates when its input buffer is ready to accept another IPMI Message via Terminal Mode. This option is typically used with automated applications that send and receive IPMI Messages using Terminal Mode. The BMC outputs the following character sequence whenever there is space for a new input message from Terminal Mode, and the 'handshake' option is enabled:

[SYS]<newline>

If a message transmission from the BMC is already in progress, the handshake sequence will be held-off until the present message transmission has completed. The BMC will typically output the handshake sequence within 1ms of the buffer space becoming available and the present message transmission (if any) completing.

13.7.8 Terminal Mode ASCII Text Commands

A small number of ASCII-text commands can be delivered while in terminal mode. The following table lists these commands. Commands are CASE SENSITIVE. *Appendix E - Terminal Mode Grammar*, lists the rules for the format of terminal mode input and output for both IPMI messages and text commands. Refer to *Table 13-13, Terminal Mode Examples* for some examples of Terminal Mode text command and IPMI messages.

Table 13-13, Terminal Mode Text Commands

Command Text	Description
SYS PWD -U USERNAME <password>	Used to activate a terminal mode session. USERNAME corresponds to the ASCII text for the username. <password> represents a printable password (up to 16 characters). If <password> is not provided, then a Null password (all binary 0's) is submitted. Passwords are case sensitive. Either the SYS PWD command (or *Activate Session* IPMI message) must be successfully executed before any command or IPMI messages will be accepted. Note that a modem connection may be automatically dropped if multiple bad passwords are entered.
SYS PWD -N <password>	-N represents a Null username. <password> represents a printable password (up to 16 characters). If <password> is not provided, then a Null password (all binary 0's) is submitted. Passwords are case sensitive. Either the SYS PWD command (or *Activate Session* IPMI message) must be successfully executed before any command or IPMI messages will be accepted. Note that a modem connection may be automatically dropped if multiple bad passwords are entered.
SYS PWD -X	-X immediately 'logs out' any presently active session. Entering an invalid password with -U or -N will also have the same effect.
SYS TMODE	Used as a 'no-op' confirm that Terminal Mode is active. BMC returns an OK response followed by "TMODE".
SYS SET BOOT XX YY ZZ AA BB	Sets the boot flags to direct a boot to the specified device following the next IPMI command or action initiated reset or power-on. XX…BB are five hex-ASCII bytes for the boot flags parameter in the Boot Options Parameters. See *Table 22-12, Boot Option Parameters.* Upon receiving this command, the BMC will also set the 'valid bit' in the boot options, and will set all the Boot Initiator Acknowledge data bits to 1b.
SYS SET BOOTOPT NN XX…NN	This is essentially a text version of the IPMI "*Set System Boot Options*" command, allows any of the boot option parameters to be set, not just the boot flags. XX…NN represents the hex-ascii for the data bytes that are passed in the Set System Boot Options request.
SYS GET BOOTOPT XX YY ZZ	This is essentially a text version of the IPMI "*Get System Boot Options*" command, allows any of the boot option parameters to be set. XX YY ZZ represents the hex-ascii for the data bytes that are passed in the Get System Boot Options request. The BMC returns the data from the command in hex-ascii format, with a maximum of four hex-

	ascii pairs per line.
SYS SET TCFG	Returns the Terminal Mode Configuration bytes where XX and YY represent hex-ascii encodings for the volatile version of data bytes 1 and 2 as specified in the Terminal Mode Configuration parameter (#29) listed in *Table 20-4, Serial/Modem Configuration Parameters,* and AA BB represent hex-ascii encoding of the non-volatile version. V:XX YY<output termination sequence> N:AA BB<output termination sequence>
SYS SET TCFG -V XX YY	This command sets the volatile Terminal Mode Configuration. XX and YY represent hex-ascii encodings for data bytes 1 and 2 as specified in the Terminal Mode Configuration parameter (#29) listed in *Table 20-4, Serial/Modem Configuration Parameters.* The BMC returns the same output as for SYS SET TCFG, above.
SYS SET TCFG -N XX YY	This command sets the non-volatile Terminal Mode Configuration. XX and YY represent hex-ascii encodings for data bytes 1 and 2 as specified in the Terminal Mode Configuration parameter (#29) listed in *Table 20-4, Serial/Modem Configuration Parameters.* The BMC returns the same output as for SYS SET TCFG, above.
SYS RESET	Directs the BMC to perform an immediate system hard reset.
SYS POWER OFF	Directs the BMC to perform an immediate system power off.
SYS POWER ON	Causes the BMC to initiate an immediate system power on
SYS HEALTH QUERY	Causes the BMC to return a high level version of the system health status in 'terse' format. The BMC returns a string with the following format if command is accepted. PWR:zzz H:xx T:xx V:xx PS:xx C:xx D:xx S:xx O:xx Where: PWR is system **P**O**W**E**R** state H is overall **H**ealth T is **T**emperature V is **V**oltage PS is **P**ower **S**upply subsystem F is cooling subsystem (**F**ans) D is Hard **D**rive / RAID Subsystem S is physical **S**ecurity O is **O**ther (OEM) zzz is: "ON", "OFF" (soft-off or mechanical off), "SLP" (sleep - used when can't distinguish sleep level), "S4", "S3", "S2", "S1", "??" (unknown) and xx is: ok, nc, cr, nr, uf, or ?? where: "ok" = OK (monitored parameters within normal operating ranges) "nc" = non-critical ('warning': hardware outside normal operating range) "cr" = critical ('fatal' :hardware exceeding specified ratings) "nr" = non-recoverable ('potential damage': system hardware in jeopardy or damaged) "uf" = unspecified fault (fault detected, but severity unspecified) "??" = status not available/unknown (typically because system power is OFF)
SYS HEALTH QUERY -V	Causes the BMC to return a high level version of the system health status in multi-line 'verbose' format. The BMC returns a string of the following format: SYS Health:xx<output termination sequence> Power: "ON", "OFF" (soft-off or mechanical off), "SLEEP" (sleep - used when can't distinguish sleep level), "S4", "S3", "S2", "S1", "Unknown" Temperature:xx<output termination sequence> Voltage:xx<output termination sequence> PowerSystem:xx<output termination sequence> Cooling:xx<output termination sequence> Drives:xx<output termination sequence> Security:xx<output termination sequence> Other:xx<output termination sequence>

	Where xx is: "OK" (monitored parameters within normal operating ranges) "Non-critical" ('warning': hardware outside normal operating range) "Critical" ('fatal' :hardware exceeding specified ratings) "Non-recoverable" ('potential damage': system hardware in jeopardy or damaged) "Unspecified fault" (fault detected, but severity unspecified) "Unknown" (status not available/unknown (typically because system power is OFF)
SYS XXXXXX yy..zz	OEM Text Commands (optional, vendor-specific). All OEM text commands are prefixed with SYS followed by XXXXXX where XXXXXX is the OEM ID expressed as a six-digit hex-ASCII number. For example, the IANA OEM IDs for Intel, HP, Dell, and NEC are 000157h (343), 00000B (11), 0002A2h (674), and 000077 (119), respectively. yy..zz represents OEM-specific text. It is recommended that OEM Text Command implementations use the same OK and ERROR completion returns be used for OEM Commands as for the IPMI-specified text commands.

13.7.9 Terminal Mode Text Command and IPMI Message Examples

The following table presents some examples of terminal mode commands and IPMI messages.

Table 13-14, Terminal Mode Examples

Console input:	**[SYS TMODE]**<crlf>	TMODE is a 'no-op' command used to confirm the BMC is operating in terminal mode.
BMC responds:	**[OK TMODE]**<crlf>	
Console input:	**[SYS PWD -U Fred letME1n]**<crlf>	User submits password for username Fred.
BMC responds:	**[OK]**<crlf>	
Console input:	**[SYS PWD -N]**<crlf>	User attempts to activate session with anonymous login (null username, null password)
BMC responds:	**[ERR CC]**	BMC returns error, e.g. 'invalid data field'.
Console input:	**[SYS RESET]<**crlf>	User resets system.
BMC responds:	**[OK]<**crlf> and resets system.	
Console input:	**[sys blah]<**crlf>	User enters an invalid command.
BMC responds:	**[ERR C1]<**crlf>	
Console input:	**[sys health query -V]<**crlf>	Verbose system health query.
BMC responds:	**[OK**<crlf>	
	Health:Critical<crlf>	
	Temperature:OK<crlf >	
	Voltages:OK<crlf>	
	Drive Subsystems:OK<crlf>	
	Power System:OK<crlf>	
	Cooling:Critical<crlf>	
	Security:OK<crlf>	
	Other:OK]<crlf>	
Console input:	**[18 xx 22]**<crlf>	IPMI *Reset Watchdog Timer* request message to BMC. xx represents the console selected sequence number and LUN field for the request.
BMC responds:	**[1C xx 22 00]**<crlf>	*Reset Watchdog Timer* response message from BMC. The same sequence number and LUN passed in the request is returned in the response.
Console input:	**[SYS 000157 My Command]**<crlf>	Submit an OEM text command
BMC responds:	**[OK 000157 My Response]**<crlf>	Get an OEM text response

13.8 Terminal Mode Line Editing

Since direct human input is likely to be used with Terminal Mode, it is useful to support a limited amount of editing to reduce the effort required to recover from the inevitable typo's that occur during text entry. Line editing is an operating mode of Terminal Mode. Line editing should be enabled when direct human entry is used, and disabled when automated entry is used.

- Line editing is enabled or disabled via an option in the serial/modem configuration parameters.
- Enabling line editing disables input time-outs.
- When line editing is enabled, echo should also be enabled.
- When line editing is enabled, the *Serial/Modem Connection Active* (Ping) message should be disabled. Otherwise, unrequested Ping messages will appear in the data stream.

- The <backspace> or <delete> key can be used to delete the last character entered.
- The <ESC> character can be used to delete the entire message. An <ESC> (1Bh) character received by the BMC immediately flushes any pending input message data. If line editing is enabled, and the <ESC> is followed by an input newline, the BMC responds by putting out an output newline sequence (typically <cr-lf>). Otherwise, the BMC just silently flushes the data and goes back to looking for a START character.
- Any illegal characters received after the START character will silently flush the message in progress. The difference between this and <ESC> is that
- Long IPMI message lines can be split across multiple lines by using a line continuation <backslash> character following immediately by the input newline sequence.
- Line continuation character support is optional for the text commands, because they're considered to be short enough to fit one line.
- Line continuation character support for OEM messages is an implementation option.

13.9 Terminal Mode Input Restrictions

The following restrictions and characteristics apply to terminal mode:

- Up to 80 printable characters are required to be supported for one line. The BMC can stop accepting new characters and stop echoing input when the 80 character limit is reached (with the exception of the <ESC>, <backspace>/<delete>, illegal character, and input <newline> characters, which will still be handled).
- The interface must support the maximum IPMI input message length that is supported on the given BMC. Per *Section 6.13, Message Size & Private Bus Transaction Size Requirements*, this will typically be 40 bytes. Since each message byte can require three input characters (two hex-ASCII digits, plus a <space> character) a 40-byte IPMI message could require 120 characters, plus the starting and ending brackets, or 122 characters, total, for the message.

13.10 Page Blackout Interval

The Page Blackout interval determines the minimum number of minutes between successive pages. The purpose for this parameter is to provide a mechanism to prevent someone from getting back-to-back pages if a flurry of events occurs. The interval applies to Dial Pages and TAP Pages. It does not apply to Dial-out PET Alerting.

The Page Blackout Interval does not turn off Platform Event Filtering or associated actions. Platform Event filtering continues while a page or blackout interval is in progress. The BMC will accumulate the set of pending actions that occur during the page and blackout interval. If an event triggers a power-down action, the page will be aborted, the power-down will occur, and the page restarted after the power down. If a reset or power-cycle action is triggered, that action will be held off until the paging process and blackout interval have concluded.

The Page Blackout Interval setting is kept in the serial/modem configuration parameters managed by the BMC.

13.11 Dial Paging

Dial Paging is accomplished by the BMC using the dialing capabilities of an external modem to connect to a paging service and enter a page using telephone numeric keypad numbers. Once a connection is established, the BMC delivers the specified Alert String from the *PEF Configuration Parameters* to the modem. The paging string directs the modem to deliver a fixed number to the paging service. The paging string is often used to deliver the phone number of the system that is generating the alert. An administrator receiving the string can then call the system with a management application and use IPMI messaging to retrieve system status, SEL entries, and other information about the alert.

13.11.1 Alert Strings for Dial Paging

Modern modems deploying the modem 'AT' command set [TIA-602] contain character options associated with the dial command that utilize built-in call detection features of the modem. Thus, the call progress detection requirements on the BMC are greatly simplified. Though the majority of modems will support all of the following options, some are not mandatory in [TIA-602]. The modem's documentation should be consulted to verify support for character options prior to configuring non-volatile dial strings. The *Alert Immediate* command can be used to test dial strings before committing them to non-volatile storage.

The following character options can be used in the Alert String for a Dial Page. These options follow the modem dial command (default ='ATD') issued by the BMC:

P	Dial using Pulse. Dialing digits after the 'P' will be sent using pulse dialing
R	Reverse Frequencies. Forces modem to dial out at the answer frequency
S=n	Dial pre-stored phone number, n
T	Dial using Tone. Dialing digits after the 'T' will be sent using touch tones
W	Wait for dial tone
@	Wait for quiet (answer)
,	Pause (2 seconds)
;	Return to command mode after dialing
!	Flash the switch hook

13.11.2 Dialing Digits

Per [TIA-602] the dialing digits consist of the ASCII characters 0..9, *, #, A, B, C, and D.

13.11.3 <Enter> Character (control-M)

The BMC recognizes the "control-M" character (0Dh) as an <ENTER> character. When the BMC encounters this character it transfers it to the modem and then delays 1 second before sending any remaining characters in the page string. Note that the BMC automatically issues an <ENTER> character after sending out the Dial String when performing a Dial Page.

13.11.4 Long Pause Character (control-L)

The BMC also recognizes the "control-L" character (0Ch) as the trigger for generating a 10-second 'long pause' sequence. When the BMC encounters this character, it doesn't send it to the modem but instead delays 10 seconds before sending any remaining characters in the page string.

13.11.5 Empty (delimiter) Character (FFh)

The BMC recognizes FFh in the page string as a 'delimiter' character used by BIOS. The character is provided as a potential aid to for interfaces, such as BIOS setup, that may split the page string into multiple fields for presentation to the user. The BMC ignores the character and does not transfer it to the modem.

13.11.6 'Null' Terminator Character (00h)

The BMC recognizes this character as a terminator for the Dial String. This terminator is used whenever the Dial String data consists of fewer characters than the maximum length for the Dial String. Note that the BMC automatically issues an <ENTER> character after sending out the Dial String when performing a Dial Page.

13.12 TAP Paging

TAP (Telocator Access Protocol) is a popular protocol for sending an alphanumeric page by connecting to a paging service using a serial modem. IPMI supports TAP as an option for delivering a short alert page to a remote paging service. This capability is referred to as TAP Paging. TAP Paging is triggered from the IPMI Platform Event Filtering capability. It can also be triggered 'manually' via the *Serial/Modem Connection Active* (Ping) command.

The protocol is described in the [TAP]. Per [TAP], there are two types of remote page entry: automated and manual. The TAP implementation for IPMI operates using the management controller as an automated entry device.

A TAP Paging transaction consists of one or more blocks. A block can contain up to 250 characters of information. Each block contains one or more short message strings that form a message sequence. Message data can span blocks. The short message strings are also referred to as 'Fields' in the TAP specification. Since only a small number of characters are delivered in an IPMI TAP Page, only a single block will be transmitted.

There are typically two fields within the first block. The first field, Field #1, is called the Pager ID field. Some paging services refer to this as the PIN (Pager ID Number). This field is used to identify the target pager. The second field, Field #2, is the alphanumeric paging message. TAP only directly supports delivery of 7-bit ASCII characters. There is an associated protocol for transmitting 8-bit characters via TAP, but that protocol is not supported in this specification.

TAP includes provision for an optional alphanumeric six-character password for the paging service. The password is also set via the serial/modem configuration parameters.

Appendix F - TAP Flow Summary, presents an additional overview and implementation notes for TAP paging via a BMC.

13.12.1 TAP Escaping (data transparency)

TAP allows ASCII control characters (00h to 20h) to be sent in the alphanumeric paging transaction as two-character sequences. TAP requires that the characters be escaped per the following table. The BMC automatically performs escaping when transmitting TAP messages.

Table 13-15, TAP Escaping

Character	value	Escaped as:	Escaping mandatory?
<EOT>	04h	1Ah, 44h	yes
<STX>	02h	1Ah, 42h	yes
<ETX>	03h	1Ah, 43h	yes
<LF>	0Ah	1Ah, 4Ah	yes
<CR>	0Dh	1Ah, 4Dh	yes
<ETB>	17h	1Ah, 57h	yes
<SUB>	1Ah	1Ah, 5Ah	yes
<ESC>	1Bh	1Ah, 5Bh	yes
Other Control Characters	-	1Ah, (value + 40h) e.g. 20h → 1Ah, 60h	optional

[TAP] states that "any [optional] control character may be made transparent at the implementor's discretion". The IPMI serial/modem configuration options for TAP paging include a control-character map similar to that used in PPP so that the user can configure which control characters get escaped for delivery to a particular TAP service.

13.12.2 TAP Checksum

TAP messages include their own checksum format. The checksum is transmitted using three printable ASCII characters. Refer to [TAP] for the checksum algorithm.

13.12.3 TAP Response Codes

Per [TAP], each handshake message that is returned from the paging service is specified to start with a 3-character response code. The values for these codes are specified in [TAP]. The implementation can optionally return a set of the last TAP Response Codes using the *Get TAP Response Codes* Command as an aid to TAP connection setup and debugging.

13.12.4 TAP Page Success Criteria

The management controller can use the TAP response codes to determine whether a page was successfully sent or not. One of the following response codes, received after the management controller sends an 'End-of-Transaction' <ETX> sequence, are used a positive indication of a successful page. Optionally, the page can be considered successful if an <ACK> is received following the end-of-transaction. The serial/modem configuration parameters include a setting that selects which of these confirmation mechanisms is used.

Table 13-16, TAP Success Codes

code	TAP Definition
211	Page(s) Sent Successfully
213	Message accepted - held for deferred delivery.

13.13 PPP Alerting

PPP Alerts are accomplished by the BMC connecting to a remote LAN via a PPP account and then delivering a UDP Datagram that contains an SNMP Trap formatted per the IPMI Platform Event Trap (PET) Format specification. Information for the PET trap comes from the Event Message that generated the alert and from the serial/modem configuration parameters for PET.

14. Event Messages

Event Messages are special messages that are sent by management controllers when they detect significant or critical system management events. This includes messages for events such as 'temperature threshold exceeded', 'voltage threshold exceeded', 'power fault', etc. The Event Message generator (the device generating an Event Message) notifies the system of the event by sending an "Event Request Message" to the Event Receiver Device.

When the Event Receiver gets a valid Event Message, it sends a response message to the generator of the Event Message. It then typically transfers the message to the System Event Log. The Event Receiver does not interpret the Event Messages it receives. Thus, new Event Message types can be added into the system without impacting the Event Receiver implementation.

In some systems, the Event Receiver will need to interrupt the system to notify it that there is an Event Message to be logged. It is desirable for the implementation to have verified and buffered Event Messages in their entirety before issuing such an interrupt. This way, the interrupt handler will not need to wait for the Event Message transmission to complete first.

14.1 Critical Events and System Event Log Restrictions

The platform's System Event Log is typically of limited size (~3 to ~8 KB, depending on implementation). Therefore, it is important to refrain from filling the System Event Log with non-critical 'clutter'.

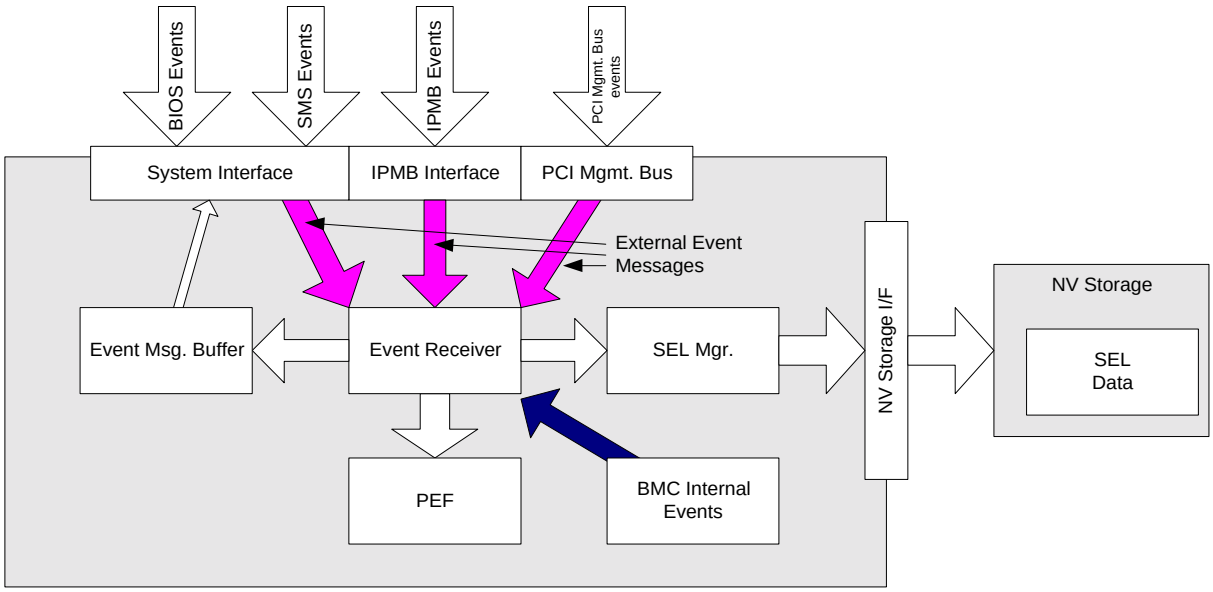
The System Event Log is primarily intended for capturing *Critical Events*. These include events that require immediate logging to guarantee that they're available for 'post-mortem' analysis, and events that may require quick system responses, such as system power off, or shutdown.

Critical events include out-of-range temperature and voltage events, hardware failures such as power supply or fan failures, interrupts and signals that affect system operation such as NMIs and PCI PERR (parity error) and SERR (system error). Critical Events also include events that impact system data integrity, such as the uncorrectable ECC errors, or system security, such as 'chassis intrusion'.

In addition to events that indicate 'failure' conditions, events that indicate impending failures are also considered to be critical events. This includes events for reaching 'warning levels' for things such as system temperature or error counts. The assertion of 'Predictive Fault' information is also considered critical, particularly if the monitored device does not have a direct 'failure' indication.

Non-critical events, such as the return to an 'OK' state from a 'Warning' state *should not be sent as critical events*. Non-critical system information is normally obtained by System Management Software polling sensors and management controllers for their status.

Table 14-1, Event Message Reception



The preceding figure presents a conceptual illustration of the manner in which Event Messages can be handled by a Baseboard Management Controller device that uses an external non-volatile storage device to hold the System Event Log.

The figure shows a BMC with a shared system messaging interface where Event Messages can be delivered from either BIOS, SMS (system management software / OS), or an SMI Handler, and an IPMB interface and through which it can receive Event Messages from the Intelligent Platform Management bus. The BMC can also generate 'internal' Event Messages.

When the BMC receives a message via the system or IPMB interfaces, a 'Message Handler' function recognizes the message as being for the 'Event' functionality in the BMC and passes the message information on to the 'Event Receiver' function. The Event Receiver function then takes the message content and issues a request to a 'SEL Mgr.' function that formats the message as an SEL Entry and calls the FLASH Interface to have the data stored.

The Event Receiver function is also responsible for driving the response message back through the messaging system. This way, message acknowledgment or error reporting can be provided.

14.2 Event Receiver Handling of Event Messages

This section presents some implementation advice for the Event Receiver device. Please refer to the *Intelligent Platform Management Bus Communications Protocol Specification* for additional information on Event Message handling.

Since retries of Event Messages are part of the IPMB protocol, there is the potential for the Critical Event Handler to receive more than one Event Messages for the same event. The Seq field allows repeated Event Messages to be discriminated from new Event Messages. Event Messages from a Event Generator that match an earlier Event Message can be ignored.

The option to disable SEL Logging only affects events that are received from the IPMB and PCI Management Bus interfaces. Devices on the IPMB and PCI Management Bus are more likely to generate events 'automatically' while the other interfaces are primarily driven by either local or remote software which is assumed to have more control as to whether it generates events or not.

It is recommended that Event Receiver keep a table or queue of the Event Messages it has received. Any new event message from the same source and of the same type, but with a different sequence number, would replace the previous entry.

There are many ways to implement such a table or queue. Any implementation should provide enough tracking support to handle previously received Event Messages for all the 'known' Event Generators in the basic system. For example, a system that has four management controllers on the IPMB that can generate Event Messages should track the previously received Event Messages from those devices.

It is desired that the Event Receiver can track at least six additional Event Generators to cover additional Event Generators that are added into the system. (One common add-on would be an emergency management. Other possible 'add-on' event generators would be other systems and peripheral boxes in a "managed cluster" arrangement).

The Event Receiver implementation should account for the possibility that there can be more different Event Generators than there are slots in the table. This can be managed by implementing the table with an 'LRU' deletion algorithm, where the oldest tracked Event Messages are deleted if a new Event Message comes in and the table or queue is full. It can be assumed that there will rarely be more than two event messages that would be in the state where they are to be re-transmitted because of a lost acknowledge.

With this type of design, the most anomalous behavior would be the multiple recording of the same event. This would only be seen under artificially generated 'stress' testing and would only be able to occur if there were more event message sources than table slots.

It is also recommended that the Event Receiver implement the 'Seq Timeout' as specified in the IPMB Communications Protocol specification.

14.3 IPMB Seq Field use in Event Messages

This section presents a review of the IPMB Seq field and the manner in which it is used when Event Messages are delivered via the IPMB.

The Event Receiver uses the Seq field to reject retried (duplicate) Event Request Messages that it may receive. The Event Generator will re-send an Event Request Message if it does not receive the Event Response Message. It is possible that the response could get corrupted, causing the Event Generator to re-send the original request even though the Event Receiver had already successfully received it. This is one way that an Event Receiver could get more than one Event Request Message for the same event. When the Event Generator re-sends the Event Request Message, it does so with the same Seq value that it used for the original try. The Event Generator will increment the Seq value the next time it has a new Event Request Message to send.

When Event Messages are delivered via the IPMB, the IPMB message's Seq field is used to allow Event Receiver to discriminate whether the Event Message is for a new occurrence of a given event, or is a re-transmission of a previous Event Message for that event. The IPMB Seq field should not be confused with being a sequence number for tracking multi-message transfers, as might be its use in other serial protocols.

If the Event Receiver receives an Event Message where the Cmd, NetFn, LUN, and Seq fields match the previous event message from the same Requester, it can assume that the latter message is a re-transmission and return a 'normal completion' (00h) as a response to valid, duplicated requests. The Event Receiver does not log duplicate events.

If the Event Receiver does not return a response, the Event Generator retries up to its retry limit count and then concludes that the Event Request failed. Event Generator devices on the IPMB do not send new Event Messages until they've finished sending the previous Event Message (including retries). This eliminates the need for the Event Receiver to maintain status for multiple Seq numbers from a single Event Generator.

The data fields for the Event Request Message *are not* included in the comparison. This is because the Event Request Message may return a data field that reflects a 'present state' or data value that could vary with each retry.

Refer to the *Intelligent Platform Management Bus v1.0 Communications Protocol Specification* for more information on the Seq field.

14.4 Event Status, Event Conditions, and Present State

A sensor tracks present state and *Event Conditions*. An Event Condition is that set of comparisons applied to the present state and previous state that produces a given *Event Status*.

A management controller typically polls for Event Conditions. When it sees a condition become active, it updates the Event Status for the sensor. The process of updating the present state Event Status is referred to as *Scanning* or *Sensor Scanning.*

The Event Status is those bits that are reported in the *Get Sensor Event Status* command. As long as scanning is enabled, the Event Status bits will be updated according to changes in Event Status. This is independent of whether Event Messages are generated on a given event. That is, turning off Event Message Generation for a particular state does not turn off scanning or updates of the Event Status.

The *Get Sensor Reading* command returns *State Bits* reflecting the *present state* of the sensor. If the sensor is an 'auto- re-arm' sensor, these bits can also represent the Event Status if hysteresis is factored in. Thus, the *Get Sensor Events* command is optional for auto- re-arm sensors. An application uses the masks in the SDR to determine which bits reflect both current state and event status, and which bits reflect current state only.

The condition that causes an Event Message to be sent is referred to as the 'Event Trigger'. The classification of a sensor indicates whether the corresponding event was discrete, or threshold-based. The sensor classification is part of the Event/Reading Type Code (see section 36.1, *Event/Reading Type Codes*).

14.5 System Software use of Sensor Scanning bits & Entity Info

System software must ignore any sensor that has the sensor scanning bit disabled - if system software didn't disable the sensor. This provides an alternate mechanism to allow the management controller to automatically adjust the sensor population without requiring a corresponding change of the sensor data records. For example, suppose the management controller has a way of automatically knowing that a particular temperature sensor will be absent in a given system configuration if a given processor is also absent. The management controller could elect to automatically disable scanning for that temperature sensor. System management software would ignore that sensor even if it was reported in the SDRs.

Note that this is an alternate mechanism that may be useful in some circumstances. The primary mechanism is to use the Entity ID information in the SDRs, and combine that information with presence detection for the entity.

If there is a presence detection sensor for a given entity, then system management software should ignore all other sensors associated with that entity. Some sensors have intrinsic support for this. For example, a sensor-specific Processor sensor has a 'Processor Presence' bit. If that bit is implemented, and the processor is absent, any other sensors and non-presence related bits associated with that processor can be ignored. If the sensor type doesn't have an intrinsic presence capability, you can implement an 'Entity Presence' sensor. This sensor solely reports whether a given Entity is present or not.

14.6 Re-arming

Re-arm refers to resetting internal device state that tracks that an event has occurred on the sensor. After a sensor is re-armed the device will re-check the event condition and re-generate the event if the event condition exists.

If the event condition already exists at the time that the re-arm is initiated, then it is possible that the event will be regenerated immediately following the conclusion of the re-arm. The delay from the re-arming of a sensor to the regeneration of the event is device implementation dependent. An *initial update in progress* bit is provided with

the *Get Sensor Reading* and *Get Sensor Event Status* commands to help software avoid getting incorrect event status due to a re-arm.

14.6.1 'Global' Re-arm

A device that receives a *Set Event Receiver* command shall 're-arm' event generation for all its internal sensors.

15. Platform Event Filtering (PEF)

Platform Event Filtering (PEF) provides a regular mechanism for configuring the BMC to take selected actions on event messages that it receives or has internally generated. These actions include operations such as system power-off, system reset, as well as triggering the generation of an Alert.

The BMC maintains an *event filter table* that is used to select which events trigger a page (or other action) and which actions to perform. Each time the BMC receives an event message (either externally or internally generated) it compares the event data against the entries in the event filter table. The BMC scans all entries in the table and collects a set of actions to be performed as determined by the entries that were matched.

Event filtering is independent of Event Logging. That is, Event Logging and Event Filtering (and associated actions) are enabled/disabled independent of one another.

15.1 Alert Policies

When an Alert is triggered via PEF the alerting process is directed by an *Alert Policy.* An alert policy is a collection of one or more alert destinations. An alert policy can support a mix of different alert destination types and channels. For example, one policy could include LAN, dial page, and TAP alerts to different locations. The destinations in a policy are processed in sequence. Whether a given destination will be used or not can be configured to be dependent on whether the alert to the previous destination was successful or not.

Alert Policy data is stored in an *Alert Policy Table* that is part of the PEF configuration parameters. An implementation can support multiple policies. A *policy number* identifies different policies in the table. The alert policy number is used in the Event Filter Entry to select what alert policy is used when a match occurs. This mechanism allows different alert policies to be associated with different classes of events. For example, one policy to be used for ‘high priority’ events and a different policy for ‘low priority’ events.

Some alerts, such as alphanumeric pages, can be associated with *Alert Strings*. The combination of Event Filter Entry and alert destination are used to select a given Alert String from a set of strings kept in the PEF configuration parameters. This enables different strings to be sent based on what event filter was matched and where the alert is being sent.

15.2 Deferred Alerts

When an alert policy is initiated, it’s possible that the communication path to the destination could already be busy processing an earlier alert. To handle this situation, the implementation internally queues up information that tracks alert policies and destinations to be processed. Alerts that have been postponed are referred to as *Deferred Alerts.*

A BMC that supports alerting is required to support deferred alert policies for at least **eight** events.

15.3 PEF Postpone Timer

Event logging takes precedence over PEF actions. That is, BMC logging of the event is completed prior to initiating any PEF actions. PEF can occur immediately after the event is logged, or it may be postponed by the PEF Postpone Timer. The PEF Postpone Timer is a separate timer that allows system software time to process events instead of PEF. PEF will occur if system software does not handle the event before the PEF Postpone Timer expires.

15.4 PEF Startup Delay

Platform Event Filtering is active whenever the BMC is in a state to receive events, either internally or externally generated. This includes events received over the system interface. Platform Event Filtering is not available when the BMC is in manufacturing test, modal SDR update, or firmware update modes.

PEF triggered actions may be postponed during certain intervals of BMC operation. The PEF Startup Delay causes Platform Event Filtering triggered power-down, reset, and power-cycle actions to be postponed when the system is either powered up or is reset **locally via a pushbutton** or other local 'front-panel' user interface. OEM actions may or may not be postponed, at the choice of the OEM implementation. Alerts are not postponed by the PEF Startup Delay. There is a separate, optional, PEF configuration parameter that can control whether Alerts are delayed on system startup. An implementation may allow the time delay for the PEF Startup Delay to be configured via the *Set PEF Configuration Parameters* command.

It is recommended that the act of entering BIOS setup automatically disables Platform Event Filtering, and that exiting BIOS setup automatically restores the prior PEF enabled/disabled state (provided that the user does not explicitly change the PEF configuration while in setup).

The combination of the delay and BIOS setup gives the user the opportunity to enter setup and disable PEF. These provisions are to allow recovery in case an incorrectly configured filter/action prevents the system from running by powering it off, power cycling, or resetting it whenever the BMC initializes. Disabling PEF must immediately cancel any pending PEF actions and deferred alerts.

15.4.1 Last Processed Event Tracking

A non-volatile 'Last Software Processed Event' storage location holds the Record ID for the last SEL Record that system software has processed. System software writes to that location to identify which records it has processed. A corresponding 'Last BMC Processed Event' value holds the Record ID for the last event in the SEL that the BMC processed. These values can be set and retrieved by software using the *Set Last Processed Event ID* and *Get Last Processed Event ID* commands, respectively.

If PEF is disabled, the Last BMC-processed Event holds the Record ID for the last event that was received. Clearing the SEL automatically clears the Last Software Processed Event and Last BMC Processed Event values.

If PEF is enabled and the BMC loses power or is reset before the PEF Postpone Timer expires, the BMC will automatically perform PEF against any existing, unprocessed events in the SEL once the BMC has restarted and reinitialized.

Once enabled, the PEF Postpone timer starts running as soon as an unprocessed event is detected in the SEL. If the SEL already contains unprocessed events, the timer will start immediately.

The timer does not automatically reset on events received while the timer is running, but is reset by system software after it sets the Last Software Processed Event value.

15.5 Event Processing When The SEL Is Full

If the SEL is full, new events will still be put into the Event Message Buffer (if the optional Event Message Buffer implemented). The Event Message Buffer for IPMI v1.5 is not overwritten if new events come in. Therefore, if the Event Message Buffer is full, further events will not go into the event message buffer until its cleared.

If PEF is implemented, events will also be accepted into a 'hidden' internal queue or buffer so they can be processed by PEF. That buffer is only required to hold a single event. Thus, if that internal buffer gets full, event messages will be rejected until a new space opens up.

If neither an Event Message Buffer nor PEF are implemented, events will be rejected by the BMC once the SEL gets full.

An implementation is allowed to provide a proprietary 'SEL Aging' option that automatically clears out SEL entries if they're more than a certain age old. If this is done, the algorithm must set the SEL 'most recent erase timestamp' to reflect the time entries were deleted. It must also be possible to configure the system to operate with the aging algorithm turned off.

15.6 PEF Actions

BMC will scan entire list, collecting a set of actions. Actions will then be executed in priority order. *An alert action can occur in combination with any other action (in priority order). The power down, power cycle, and reset actions are mutually exclusive.* If a combination of power down, power cycle, and/or reset actions results, only the highest priority action will be taken.

Table 15-1, PEF Action Priorities

Action	Priority	Additional Information
power down	1	(optional)
power cycle	2	(optional) Will not be executed if a power down action was also selected.
reset	3	(mandatory) Will not be executed if a power down or power cycle action was also selected.
Diagnostic Interrupt	4	(optional) The diagnostic interrupt will not occur if a higher priority action is also selected to occur.
Send Alert	5	(mandatory if alerting is supported) Send alerts in order based on the selected Alert Policy. Alert actions will be deferred until after the power down has completed. There is an additional prioritization within alerts being sent: based on the Alert Policy Table entries for the alert. This is described further in *Section 15.11, Alert Policy Table*.
OEM	OEM	(optional) Priority determined by OEM.

15.7 Event Filter Table

The Event Filter Table consists of a set of rows or 'entries' that define each filter. The following table specifies the fields that comprise a row in the Event Filter Table. These entries include a series of masks that the BMC applies to the event data. The fields are designed such that a combination of absolute and 'wildcarded' comparisons can be used for matching fields in the event record. Thus, either a single event or multiple events can match up with a single Event Filter Table entry.

A PEF implementation is recommended to provide at least 16 entries in the event filter table. A subset of these entries should be pre-configured for common system failure events, such as over-temperature, power system failure, fan failure events, etc. Remaining entries can be made available for 'OEM' or System Management Software configured events. Note that individual entries can be tagged as being reserved for system use - so this ratio of pre-configured entries to run-time configurable entries can be reallocated if necessary.

A match occurs when there are event filter table matches (exact or wild-carded) for all compared fields in the event message.

There are two things that can kick off PEF: the arrival of a new event or BMC startup with pending events.

Table 15-2, Event Filter Table Entry

Byte	Field	Description
1	Filter Configuration	[7] - 1b = enable filter 0b = disable filter [6:5] - 11b = reserved 10b = manufacturer pre-configured filter. The filter entry has been configured by the system integrator and should not be altered by software. Software is allowed to enable or disable the filter, however. 01b = reserved 00b = software configurable filter. The filter entry is available for configuration by system management software. [4:0] - reserved
2	Event Filter Action	All actions are optional for an implementation, with the exception of Alert which is mandatory if alerting is supported for one or more channels. The BMC will return 0b for unsupported actions. Software can test for which actions are supported by writing 1's to the specified fields and reading back the result. (Note that reserved bits must be written with 0's) [7:6] - reserved [5] - 1b = Diagnostic Interrupt (NMI) 0b = no Diagnostic Interrupt [4] - 1b = OEM action 0b = no OEM [3] - 1b = power cycle 0b = no power cycle [2] - 1b = reset 0b = no reset [1] - 1b = power off 0b = no power off [0] - 1b = Alert 0b = no Alert
3	Alert Policy Number	Used to select an alerting policy set from the Alert Policy Table. The Alert Policy Table holds different policies that configure the order in which different alert destinations and alerting media are tried. [7:4] - reserved [3:0] - policy number. Value is 'don't care' if Alert is not selected in the Event Filter Action.
4	Event Severity	This field can be used to fill in the Event Severity field in a PET alert. The severity values are based on the 'DMI' severity values used for the generic sensor event/reading type code. In the case that more than one event filter match occurs for a given Alert Policy Number, the numerically highest severity value will be used. 00h = unspecified 01h = Monitor 00 0001 02h = Information 00 0010 04h = OK (return to OK condition) 00 0100 08h = Non-critical condition 00 1000 a.k.a. 'warning' 10h = Critical condition 01 0000 20h = Non-recoverable condition 10 0000
5	Generator ID Byte 1	Slave Address or Software ID from Event Message. FFh = match any
6	Generator ID Byte 2	Channel Number / LUN to match. FFh = match any see section *26, SEL Record Formats.*
7	Sensor Type	Type of sensor. FFh = match any
8	Sensor #	FFh = match any
9	Event Trigger (Event/Reading Type)	FFh = match any

10, 11	Event Data 1 Event Offset Mask	This bit field is used to match different values of the least significant **nibble** of the Event Data 1 field. This enables a filter to provide a match on multiple event offset values. Bit positions 15 through 0 correspond to the offset values Fh - 0h, respectively. A 1 in a given bit position will cause a match if the value in bits 3:0 of the Event Data 1 hold the corresponding value for the bit position. Multiple mask bits can be set to 1, enabling a match to multiple values. A match must be made with this field in order to have a match for the filter. data 1 7:0 - mask bit positions 7 to 0, respectively. data 2 15:8 - mask bit positions 15 to 8, respectively.
12	Event Data 1 AND Mask	This value is applied to the entire Event Data 1 byte. The field is Used to indicate 'wildcarded' or 'compared' bits. This field must be used in conjunction with Compare 2. To match any Event Data field value, just set the corresponding AND Mask, Compare 1, and Compare 2 fields to 00h. (See *Section 15.8, Event Data 1 Event Offset Mask* for more information). Note that the Event Data 1 AND mask, Compare 1 mask, and Compare 2 masks will typically be set to wild-card the least significant of Event Data 1 in order to allow the Event Data 1 Event Mask field to determine matches to the event offset. Bits 7:0 all have the following definition: 0 = Wildcard bit. (drops this bit position in the Event Data byte out of the comparison process) Corresponding bit position must be a 1 in Compare 1, and a 0 in Compare 2. (Note - setting a 0 in this bit, a 1 and Compare 1 and a 1 in Compare 2 guarantees that you'll *never* have a match.) 1 = use bit for further 'exact' or 'non-exact' comparisons based on Compare 1 and Compare 2 values.
13	Event Data 1 Compare 1	Used to indicate whether each bit position's comparison is an exact comparison or not. (See *Section 15.8, Event Data 1 Event Offset Mask* for more information). Here, 'test value' refers to the Event Data value *after* the AND Mask has been applied. Bits 7:0 all have the following definition: 1 = match bit in test value exactly to correspond bit position in Compare 2 0 = contributes to match if corresponding bit in test value matches corresponding bit in Compare 2.
14	Event Data 1 Compare 2	(See *Section 15.8, Event Data 1 Event Offset Mask* for more information). Here, 'test value' refers to the Event Data value *after* the AND Mask has been applied. Bits 7:0 all have the following definition: 1 = match a '1' in corresponding bit position in test value. 0 = match a '0' in corresponding bit position in test value.
15	Event Data 2 AND Mask	
16	Event Data 2 Compare 1	
17	Event Data 2 Compare 2	
18	Event Data 3 AND Mask	
19	Event Data 3 Compare 1	
20	Event Data 3 Compare 2	

15.8 Event Data 1 Event Offset Mask

The *Event Data 1 Event Offset Mask* field in the Event Filter is used to match multiple bits in the Event Offset field of the Event Data 1 byte of an event. The least significant nibble of event data 1 typically holds an event offset value. This offset selects among different possible events for a sensor. For example, a 'button' sensor supports a set of sensor-specific event offsets: 0 for Power Button pressed, 1 for Sleep Button pressed, and 2 for Reset Button pressed. When an event is generated, it could have a 0, 1, or 2 in the event offset field depending on what button press occurred.

The Event Offset Mask makes it simple to have a filter match a subset of the possible event offset values. Each bit in the mask corresponds to a different offset values starting with bit 0 in the mask corresponding to offset 0. For example, if it is desired to have a filter match offsets 0 and 2, but not 1, the mask would be configured to 000_0000_0000_0101b.

15.9 Using the Mask and Compare Fields

The AND Mask and the Compare 1 and Compare 2 fields are used in combination to allow wildcarding, 'one or more bit(s)', and exact comparisons to be made between bits in the corresponding event data byte. One way to understanding the bits is to look at the way they're used in combination. First the AND Mask is applied to the test value. The result, referred to below as the *test value*, is then bit-wise matched based on the values in the Compare 1 and Compare 2 fields, as summarized in the following table.

Table 15-3, Comparison-type Selection according to Compare Field bits

Compare 1	Compare 2	comparison	description
1	1	exact compare to 1	This bit in the test value must be = 1 for a match. If it's 0, there is no match. All 'exact' comparison bits must match the corresponding bits in the test value for a match.
1	0	exact compare to 0	This bit in the test value must be = 0 for a match. If it's 1, there is no match. All 'exact' comparison bits must match the corresponding bits in the test value for a match.
0	1	'non exact' compare to 1	If this bit in the test value is 1, it contributes to a match. There will be a match if any one of the bit positions that has a '0' in Compare 1 field has a bit in the test value that matches the polarity given in the corresponding bit position in the Compare 2 field. - unless there are exact comparisons that don't match.
0	0	'non exact' compare to 0	If this bit in the test value is 0, it contributes to a match. There will be a match if any one of the bit positions that has a '0' in Compare 1 field has a bit in the test value that matches the polarity given in the corresponding bit position in the Compare 2 field. - unless there are exact comparisons that don't match.

15.10 Mask and Compare Field Examples

The following examples show how the fields are used. See *Appendix B - Example PEF Mask Compare Algorithm* for example matching algorithm.

Example 1:		Match (bit 2 = 1) AND (bit 1 = 1), and ignore all other bits.
AND Mask:	0000 0110	Force all bits except bits 2 and 1 to 0. (Forcing to 0 and comparing exactly to 0 makes the other bits 'don't care')
Compare 1:	1111 1111	Compare all bits exactly.
Compare 2:	0000 0110	Compare for bits 2 and 1 both = 1, and remaining bits = 0.

Example 2:		Match (bit 2 = 1) OR (bit 1=1), and ignore all other bits.
AND Mask:	0000 0110	Force all bits except bits 2 and 1 to 0.
Compare 1:	1111 1001	Compare for at least one of bit 2 or bit 1being polarity specified in the corresponding bit position in Compare 2. Compare remaining bits exactly.
Compare 2:	0000 0110	Compare for bit 2 or bit 1 = 1, and remaining bits = 0 exactly.
Example 3:		Match (bit 2 = 1) AND (bit 1 = 0)
AND Mask:	0000 0110	Force all bits except bits 2 and 1 to 0
Compare 1:	1111 1111	Compare all bits (after AND) exactly
Compare 2:	0000 0100	Compare for bit 2 = 1 and bit 1 = 0, and remaining bits = 0 exactly.
Example 4:		Match bit 2 = 1 OR bit 1 = 0
AND Mask:	0000 0110	Force all bits except bits 2 and 1 to 0.
Compare 1:	1111 1001	Compare all bits except bits 2 and 1 exactly.
Compare 2:	0000 0100	Compare for bit 2 = 1 OR bit 1 = 0, and remaining bits = 0 exactly.
Example 5:		Match most significant nibble = 1010 exactly, and any bit in LSN = 1.
AND Mask:	1111 1111	Look at all bits
Compare 1:	1111 0000	Compare all bits in MSN exactly.
Compare 2:	1010 1111	Compare MSN = 1010 exactly, and for a 1 in one or more positions in LSN
Example 6:		match MSN = 1010 exactly, and any bit in LSN = 0.
AND Mask:	1111 1111	Look at all bits
Compare 1:	1111 0000	Compare all bits in MSN exactly.
Compare 2:	1010 0000	Compare MSN = 1010 exactly, and for a 0 in one or more positions in LSN

15.11 Alert Policy Table

Platform Event Filtering supports alerting as one of the selectable actions that can occur when an event matches an event filter table entry. The alerting media and the different alert destinations that are tried are determined by the settings in the Alert Policy Table.

The Alert Policy Table definition enables implementations to offer multiple policy sets. For example, one policy could be configured to generate LAN Alerts and Pages and be associated with critical and non-recoverable events, while another may generate only LAN Alerts and be associated with non-critical events. It would even be possible to configure a ‘Chassis Security’ event to notify one party, while an ‘Over-temperature’ event could be delivered to a different destination.

The table entry also contains a parameter that can be used to select whether the alert is delivered to all enabled destinations, or that different destinations are tried until the alert is successfully delivered. Altering the policy table entries can change the whether certain destinations are processed or not.

A *policy number* is used to group multiple table entries into a *policy set*. The collection of entries determines which different media and alert types can be tried when an alert action occurs. When a Platform Event Filter table entry is configured to perform an alert action on an event, an alert policy number is also configured for the action. The BMC takes this policy number and uses it to look up the entries for the corresponding policy set in the Alert Policy Table.

The policy number also determines the priority of alert policies. *Only one starting Alert policy will be used for a given event.* If an event matches more than one alert policy, the policy with the lowest number will be used.

Implementations that support alerting are required to provide at least one table entry. It is recommended that an implementation supports at least one policy table entry for each different channel over which an alert can be delivered.

Table 15-4, Alert Policy Table Entry

Byte	Field	Description
		DATA BYTES
1	Policy Number / Policy	This value identifies the entries belonging to a particular policy set. When an Alert Action is taken, the BMC will scan the Alert Policy Table and will attempt to generate alerts based on the entries that form the policy set. [7:4] - policy number. 1 based. 0000b = reserved. [3] - 0b = this entry is disabled. Skip to next entry in policy, if any. 1b = this entry is enabled. [2:0] - policy 0h = always send alert to this destination. 1h = if alert to previous destination was successful, do not send alert to this destination. Proceed to next entry in this policy set. 2h = if alert to previous destination was successful, do not send alert to this destination. Do not process any more entries in this policy set. 3h = if alert to previous destination was successful, do not send alert to this destination. Proceed to next entry in this policy set that is to a different channel. 4h = if alert to previous destination was successful, do not send alert to this destination. Proceed to next entry in this policy set that is to a different destination type.
2	Channel / Destination	Channel that the alert is to be sent over. Channel determines which set of destination addresses or phone numbers is used. Destination addresses and/or phone numbers are set via the LAN and/or serial/modem configuration parameter commands. The Alert Type (e.g. PET, TAP, Dial Page, etc.) is specified in the configuration parameters associated with the specified destination. [7:4] = Channel Number. [3:0] = Destination selector.
3	Alert String Key	This field holds information that is used to look up the Alert String to send for this Alert Policy entry. 00h = no alert string. [7] - Event-specific Alert String 1b = Alert String look-up is event specific. The following Alert String Set / Selector sub-field is interpreted as an *Alert String Set Number* that is used in conjunction with the Event Filter Number to lookup the Alert String from the PEF Configuration Parameters. 0b = Alert String is not event specific. The following Alert String Set / Selector sub-field is interpreted as an *Alert String Selector* that provides a direct pointer to the desired Alert String from the PEF Configuration Parameters. [6:0] - Alert String Set / Selector. This value identifies one or more Alert Strings in the Alert String table. When used as an *Alert String Set Number,* it is used in conjunction with the Event Filter Number to uniquely identify an Alert String. When used as an *Alert String Selector* it directly selects an Alert String from the PEF Configuration Parameters. The Alert String Key and lookup mechanism allows the Alert String to be 'Event Specific' - meaning the string selection is determined by both the Event Policy Entry and Event Filter, or, the string can be selected by the Event Policy alone. An Alert String can be pointed to by multiple policy entries. The Alert Policy Entry identifies a particular channel and destination for an alert. This in turn, identifies the alert type. Thus, the binding of an Alert Policy Entry and an Alert String effectively provides a mechanism for allowing different Alert Strings to be selected based on the alert destination, or the type of alert destination. For example, a single Alert String could be shared among all Alert Policy Entries for 'Dial Page' destinations, while event-specific Alert Strings could be used for alerts to LAN destinations.

15.12 Alert Testing

BMC includes an ability to test alert configurations. This is accomplished through the *Alert Immediate* command. This command allows a particular alert destination to be selected and an alert sent to it. Typically, software will use the volatile settings in the configuration parameters for the channel to hold alert destination information until the setting is verified by the user, at which time it can be moved into one of the non-volatile positions.

15.13 Alert Processing

The BMC starts from the beginning of the Alert Policy Table, scanning for entries that match the event. The BMC will scan all entries in the Event Filter table and initiate the highest priority Alert Policy for which there was a match. If multiple filters match and have the same alert policy priority, the first matched event filter will be used. (Since an Alert String can be associated with an event filter, it may be important to order the event filter table such that the more 'granular' filters occupy the earlier entries, and the more generic filters occupy the later entries.)

BMC implementations are allowed to have multiple alerts simultaneously in progress on the same channel or across multiple channels.

If an alert was successfully delivered, the BMC will either stop processing the policy, or will continue to process alert policy table entries - based on whether the destination was configured to stop alert processing on success or not. Each entry in the alert policy is processed in order until all entries have been processed (meaning the alert was either sent, failed, or the alert was skipped).

15.13.1 Alert Processing after Power Loss

It is possible that more events will come in while an alert is being processed. Each time a SEL Record is completely processed for alerts, the BMC saves a copy of the Last BMC Processed Record ID to NV storage. ("Completely processed" means that there are no incompletely processed alert policies for the event). That way, if AC power is lost the BMC can tell whether there are events that may not have been processed for alerting by comparing the Last BMC Processed Record ID with the Record ID of the last SEL Entry.

It's possible that alerts were sent to some destinations, but not all, when power was lost. When power comes back up, the BMC will start processing the entire record and as a result some alerts may get re-sent.

15.13.2 Processing non-Alert Actions after Power Loss

After the BMC powers up, and before restoring the system power state, it uses PEF to check pending events for matches that would have yielded a 'power off' action. If there is a match, it sets the previous power Restore State to Off and leaves the system powered off. The BMC skips processing reset or power cycle actions that were pending when AC was lost.

In order to know how many events to process, the BMC should copy the Record ID or timestamp of the last SEL Entry and only process records up to that point as events that were pending when AC was lost. That way, if a new event comes in, it will be handled as a new event rather than as an event that was pending when AC was lost.

15.13.3 Alert Processing when IPMI Messaging is in Progress

All alerts are deferred if IPMI Messaging is in progress on the channel. The remote console software can direct the BMC to skip processing deferred events by setting the Last BMC Processed Record ID value for all filters to the Record ID of the last SEL Record.

15.13.4 Sending Multiple Alerts On One Call

To avoid unnecessary phone calls, it is desirable to have multiple alerts be delivered to a given PPP Account before hanging up, rather than hanging-up after each event. The serial/modem configuration parameters and the Alert Policy entries can be configured to support this. To have this accomplished, the configuration must fit the following rules.

- The *Connection Hold Time* parameter for the PPP Accounts should be set to a value that covers the time that you'd like the call to be maintained waiting for the next event to occur. Since a new alert will restart the

connection time time-out, the value should be set to the time required to maintain the connection *between* alerts.

- All event policies should be configured to have alerts to each channel delivered in priority order starting with the highest priority destination first.

15.13.5 Serial/Modem Alert Processing

Alerts on the same given serial/modem channel processed according to priority. Alerts to PPP Accounts are processed with higher priority than Dial Page or TAP Page alerts. PPP Accounts are prioritized according to the account selection, with the lowest account selector corresponding to the highest priority with the lowest account selector as summarized in the following table:

Table 15-5, Serial/Modem Alert Destination Priorities

Destination Type	Priority (0 = highest)	Comments
PPP Account #1	1	All destinations *behind* a given PPP Account are at equal priority. Destinations behind an account are handled in the order that they occur in the Alert Policy Table entry associated with the account.
PPP Account #2	2	
PPP Account #N	N	
Dial Page and TAP Page	N+1	All Dial Page and TAP Pages are at equal priority. They are handled in the order that they occur in the Alert Policy Table entry.

The following specifies how serial/modem alerts from the same channel are handled:

- The BMC will not prematurely terminate a PPP Alert, Dial Page, or TAP Page in progress to a given destination, with the exception that an implementation is allowed to in order to handle a power off, power cycle, or system reset transition. In this case, the alert should be resumed once the transition has completed.
- The BMC checks the event filters for matches to the event. If there is more than one alert policy selected by the match, the BMC will only execute the highest priority (lowest numbered) alert policy.
- The BMC must keep track of how far it has processed the alert policy associated with the event. It does this in case it needs to prematurely terminate a call because of a power off, power cycle, or system reset operation.
- The BMC processes each alert policy through to completion. Completion of processing means that all alerts were either sent or deferred.
- If there is no presently active connection, a connection will be made for the first alert destination in the alert policy sent and the alert sent.
- The PPP Account *Connection Hold Time* parameter determines how long a PPP call will be held open waiting for another alert. The PPP connection will only close when the time expires, or if an alert to a higher priority PPP Account occurs.
- The call to a PPP Account will be dropped without waiting for the Connection Hold Time to expire if alerts fail to all destinations for a given policy.
- The Connection Hold Time time-out must be restarted whenever a new alert is sent to the account.
- If a PPP account connection is already active and the alert is to a destination behind that PPP account, the BMC will wait for any alerts in progress to that account to complete and then send the present alert.
- If the alert is to a destination that is behind a higher priority PPP account, the present connection will be terminated as soon as the present alert to that account has completed, regardless of the connection hold time. The BMC will then dial the higher priority account and sent the alert.

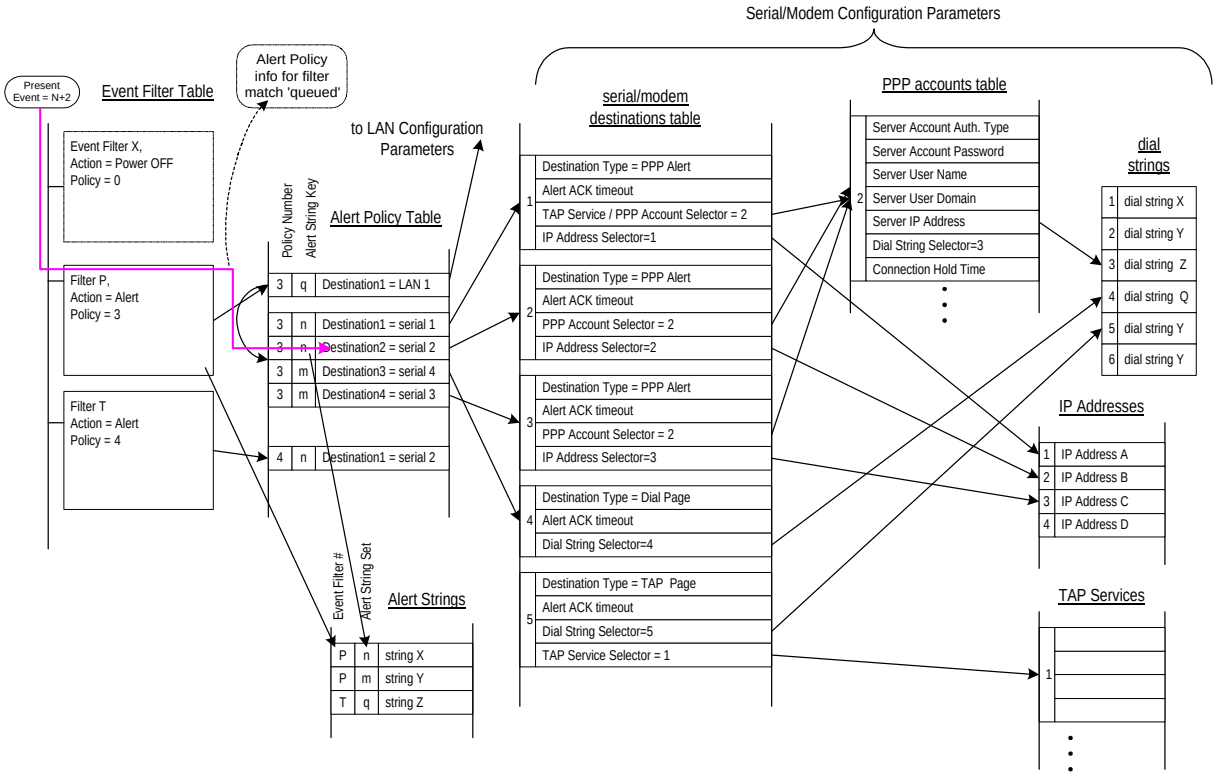
- If a PPP account connection is already open it will be used unless the alert is to a lower-priority destination type, in which case the alert will be deferred until connection hold time for the present connection expires.
- If a Dial Page or TAP Page is already active, the BMC will wait for the alert in progress to complete and then send the present alert, regardless of whether the present alert is of higher priority. (Completion of an alert in progress for an unacknowledged alert means that the alert has been sent. For an acknowledged alert, completion means the alert has been sent and the acknowledge received or all retries and timeouts have concluded and the alert is considered to have failed.)
- The Page Blackout Interval is essentially a 'throttle' that prevents pages from being sent 'back-to-back'. See *13.10, Page Blackout Interval*.

15.14 PEF and Alert Handling Example

The following figure presents a snapshot of event and alert processing using PEF. It also helps illustrate the relationship between entries in the serial/modem configuration parameters.

1. The present event being processed is identified as event with Record ID = "N+2"
2. The BMC scans all filter table entries for matches to the present event. When done, it finds three entries with Alert actions that have been matched: event filters X, P, and T.
3. The BMC handles matched filters in priority order based on the action associated with the filter. The Power Off action is higher priority than Alert actions, so filter X is acted on immediately and the power OFF action performed.
4. There are two matched filters left, both with Alert actions. Filter P is acted on because it has the lower policy number. The BMC 'queues' information for the alert policy and the event filter so it can be processed later.
5. The event triggers Alert Policy #3. The BMC sends the alert to LAN destination #1, and then sends the alert to serial/modem destination #1. The figure shows that serial/modem destination #1 is a PPP Alert destination, therefore the BMC looks up the corresponding PPP Account information from the serial/modem configuration parameters. The PPP account is account #1. This is the highest priority account. If the account is not already active, the BMC will terminate any lower priority call in progress and then call the account #1 and send the alert.
6. The completion of the alert policy for event N+2 may cause the Last BMC Processed Record ID to be set to N+2 - but it also may not. Whether the Last BMC Processed Record ID is advanced is based on whether all deferred alerts have been processed. Due to prioritization, it's possible that in some cases the alert policy triggered for event N+2 could complete while an alert policy associated with an earlier event 'N' could still have destinations to be processed.

Figure 15-1, Alert Processing Example



15.15 Event Filter, Policy, Destination, and String Relationships

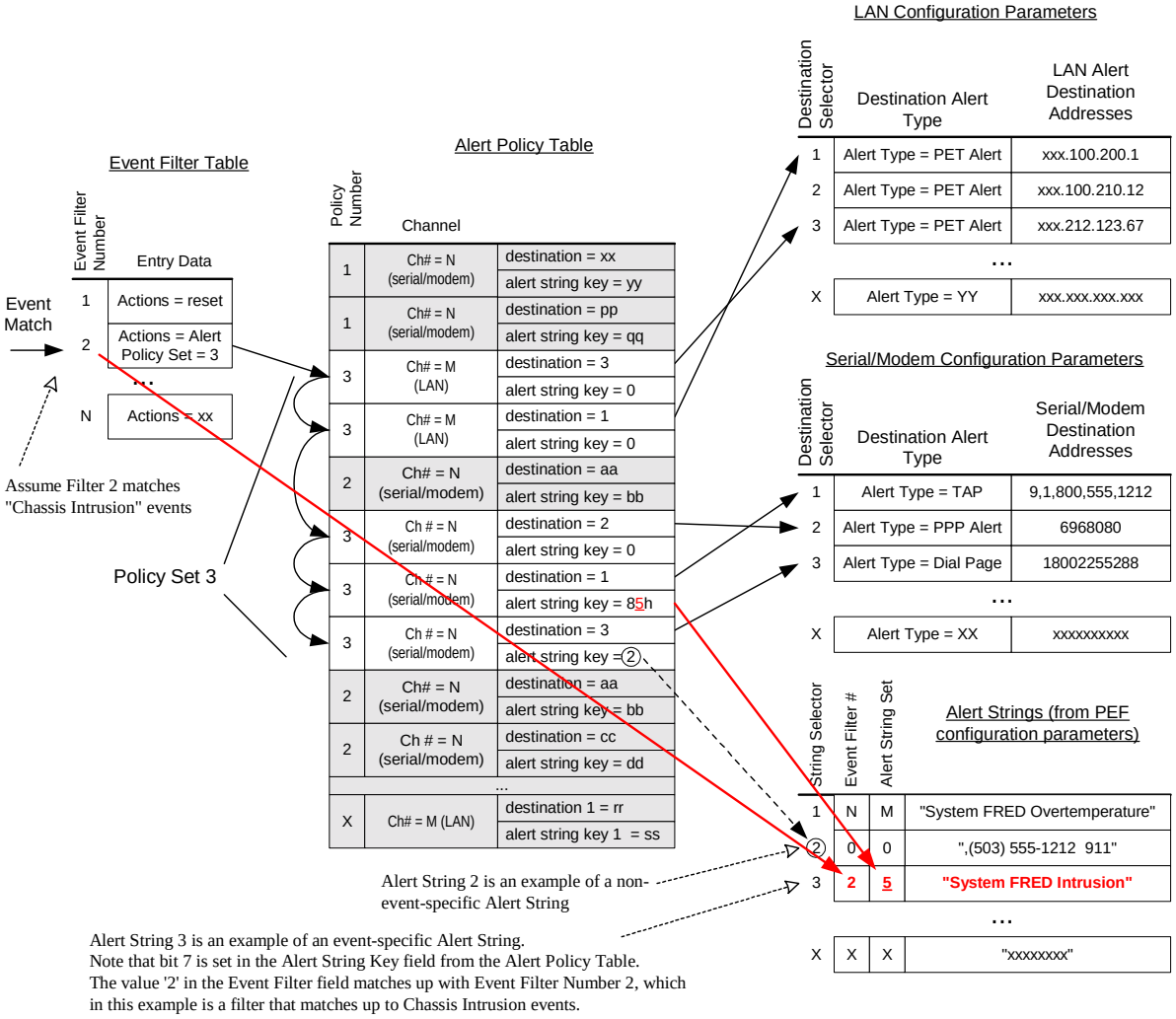
The following figure illustrates the relationship between the different structures and configuration parameters related to platform event filtering and alerting. Note that the number of table entries and support for different alert types is implementation dependent.

The figure shows the lookup process that occurs when an event matches an Event Filter Table entry. In this example, the entry triggers an Alert that activates alert policy number 3. The policy table is scanned for the first entry with a matching policy number. The first matching entry is for a LAN channel. First, a LAN PET trap to xxx.212.123.67, then, if that fails a LAN PET trap to xxx.100.200.1 will be tried. If that fails, the next matching policy entry will be tried.

The next policy is for a serial/modem channel. The first alert on this channel will be a Dial Out PET Alert (a.k.a. PPP Alert). The second attempt will be a TAP Page. Note that a short ASCII Alert String “System FRED Intrusion” is selected by the Alert Policy Entry for the TAP page. The third attempt will be a dial page. Since a dial-page is restricted to submitting ‘key pad’ digits via the modem command set, we see that the final attempt is used to deliver a numeric page with the number of the system in trouble, and a user-defined ‘911’ to indicate that there’s a serious condition.

In order to simplify the figure, certain parameters associated with the Alert destinations have been left out. For example, there are destination phone numbers and modem init strings associated with the paging destinations, and MAC address and Gateway Address values associated with the LAN Alert destinations.

Figure 15-2, Event Filter, Alert Policy, and Alert Destination, & String Relationships



15.16 Populating a PET

The following table outlines the way PET fields are populated for an IPMI alert. See [PET] for more information. The **Community String** portion of the PET can be obtained from the configuration parameters associated with the channel from which the trap is issued. If the Community String parameter is not supported, the string ‘public’ should be sent.

The following table lists the population of the PET Specific Trap fields for an IPMI alert:

Table 15-6, PET Specific Trap Fields

PET Field	IPMI Source
Event Sensor Type	Sensor Type code from event message
Event Type	Event/Reading Type code from event message
Event Offset	[7] = Event Dir bit from event message [3:0] = Event Offset for Event Data 1 byte of event message

The following table lists the IPMI source and formatting for fields that go into the 'variable bindings' fields of a PET.

Table 15-7 - PET Variable Bindings Field

PET Field	size/ type	IPMI Source
GUID	16 bytes	Recommended that BMC populate this with the System GUID. If a system GUID is not available, a device GUID for the BMC may be substituted.
Sequence # / Cookie	word	BMC should increment the value in this field for each new PET issued, but leave the value unchanged for PET retries.
Local Timestamp	dword	BMC should populate this field with the time value that would be used to log the event in the SEL. Per PET, this needs to be converted to represent number of seconds from 0:00 1/1/98. 0000 0000 = unspecified.
UTC Offset	word	Optional. UTC Offset in minutes (two's complement, signed. -720 to +720, 0xFFFF=unspecified)
Trap Source Type	byte	Class of the device or software that originated the trap on the network. Use 20h for PETs that are issued from the BMC.
Event Source Type	byte	Use 20h for events that are automatically generated by the BMC (e.g. by PEF) It is recommended that 21h be used for IPMI-format PETs that are generated by system software instead of automatically by the BMC.
Event Severity	byte	Severity (based on DMI Event Severity). If PEF specifies an event severity for the event filter that triggered the Alert, that severity should be used instead. 0x00 = unspecified 0x01 = Monitor 00_0001b 0x02 = Information 00_0010b 0x04 = OK (return to OK condition) 00_0100b 0x08 = Non-critical condition 00_1000b a.k.a. 'warning' 0x10 = Critical condition 01_0000b 0x20 = Non-recoverable condition 10_0000b
Sensor Device	byte	In IPMI this holds an ID (I^2C address or SWID) of the controller or software entity that generated the event. This comes from the event message. I.e. if the BMC received the event from controller C2h, this value would be set to C2h, *not* to the BMC's address.
Sensor Number	byte	Sensor number from the event message.
Entity	byte	Entity ID from IPMI v1.5specification. (Optional). An implementation can elect to look up the Entity ID associated with the sensor and send that information in the PET. 00h = unspecified
Entity Instance	byte	This field can hold is the Entity Instance value associated with the preceding Entity field. 00h = unspecified
Event Data	octet string (8)	Additional parametric data byte - formatted as specified by Event Type in combination with Event Source. Interpreted as individual octet fields. Event Data 1 - Populate this field with the Event Data 1 byte from the Event Message Event Data 2 - Populate this field with the Event Data 2 byte from the Event Message Event Data 3 - Populate this field with the Event Data 3 byte from the Event Message Event Data 4:8 - These bytes are not used with IPMI messages. They should be set to 00h. Software should ignore their content.
Language Code	byte	Per IPMI v1.0 FRU Information Format. FFh = 'unspecified'. This field can be used in conjunction with the OEM fields, below, to indicate the language that any strings are in. Note that language is different than character set. Character sets are specified as ASCII or UNICODE, per type/length bytes.
Manufacturer ID	dword	Manufacturer ID using Private Enterprise IDs per IANA. This should reflect the

PET Field	size/ type	IPMI Source
		ID of the manufacturer of the System from which the alert it being issued.
System ID	word	Specified by manufacturer given by Manufacturer ID field, this number can be used to identify the particular system/product model or type.
OEM Custom Fields	octet string (max. 64)	One or more fields given in IPMI v1.0 FRU Information field format: Type/length code byte followed by N data bytes for each field. Fields end when type/length byte indicates 'no more records' (C1h). A C1h in octet 47 indicates no OEM Custom Fields.

15.16.1 OEM Custom Fields and Text Alert Strings for IPMI v1.5 PET

An IPMI format PET (PET Event Source = 20h or 21h) provides additional specification on the use of the Type/Length Byte in the OEM Custom Fields by defining additional special values as follows:

PET OEM Field Type/Length Byte special values

00h → reserved
40h → reserved
80h → typed field ('PET multirecord' field)
C0h → empty field
C1h → end of fields

With this encoding, a Type/Length value of 80h indicates the start of a 'PET multirecord' field where the field format is as follows. This format enables the PET trap to carry an Alert String from PEF, while allowing OEM data to coexist in the custom fields as well.

Table 15-8, IPMI PET Multirecord Field Format

byte	Field	Definition
1	Type/Length	80h (PET multirecord field)
2	Encoding/Length	7:6 Record Encoding 00b = binary / unspecified 01b = ASCII 10b = UNICODE 11b = reserved 5:0 Record Data Length in bytes (number of bytes in Record Data field)
3	Record Type	7:4 reserved 3:0 Record Type 0h = reserved 1h = Text Alert String 2h = OEM Data per OEM Identified by IANA in first three bytes of Record Data 3h = OEM Data per OEM Identified by Manufacturer ID field
4:N	Record Data	Data per Record Type

15.17 PEF Performance Target

Excluding PEF Action Delays, a PEF action should nominally occur within **two seconds** of the corresponding Event Message being received by the BMC. For events generated internal to the BMC, this should occur within **two seconds** of the event being written to the SEL or delivered to the Event Message Buffer. Note that this does not include delays for event detection and formatting the event record, nor the time to poll and accumulate the data that triggers the event. For example, it may take the BMC several seconds to detect a fan failure, that time is *not* included in this performance target.

For alerts, this target also represents the time to *initiate* the processing of an alert policy. The actual time it takes to complete an alert policy is widely variable, dependent on factors such as whether the alert is deferred, plus elements such as telephone system and network response delays, thus a performance target is not specified for alert completion at this time.

16. Command Specification Information

This section provides specifications for elements that apply to all requests and responses presented later in this document.

16.1 Specification of Completion Codes

Completion codes are specified in *Section 5.2, Completion Codes*. Additional command-specific completion codes, if any are listed in the 'completion code' field description for the command. In some cases, use of certain command-specific completion codes is mandatory. This will be listed alongside the description of the completion code in the command table. If no command-specific completion codes are listed, the description will solely indicate that the field is the 'completion code' field. Note that the generic completion code values can be used with any command, regardless of whether additional command-specific completion codes are defined. Therefore generic completion codes are not explicitly listed in the command tables. Refer to *Section 5.2* for additional requirements and guidelines.

16.2 Handling 'Reserved' Bits and Fields

Unless otherwise noted, Reserved bits and fields in commands (request messages) and responses shall be written as '0'. Applications must ignore the state of reserved bits when they are read.

16.3 Logical Unit Numbers (LUNs) for Commands

Unless otherwise specified, commands that are listed as mandatory must be accessed via LUN 00b. An implementation may elect to make any command available on any LUN or channel as long as it does not conflict with other requirements in this specification.

16.4 Command Table Notation

The following section includes *command tables* that list the data that is included in a request or a response for each command. The completion code for a response is included as the first byte of the response data field for each command. The Network Function (NetFn) and command byte values for each command are specified in separate tables.

The following notation is used in the command tables.

Request Data — Identifies portion of the table that lists the fields that are included in the data portion of a request message for the given command.

Response Data — Identifies portion of the table that lists the fields that are included in the data portion of a response message for the given command. Note that the completion code is always listed as the first byte in the response data field.

- — **Empty Field.** A dash (-) in the *byte* column indicates that there is no request data for the command.

4 — **Single Byte Field.** A single value in the *byte* column of a command table is used to identify a single byte field. The value represents the offset to the field within the data portion of the message. In some cases a single byte field with follow a variable length field (see following), in which case the single byte offset will be represented with an alphabetic variable and number

representing the single byte field's location relative to the end of the variable length field. E.g. N+1.

5:7 **Multi-byte Field.** Indicates a multi-byte field. The *byte* column indicates the byte offset(s) for a given field. For a multi-byte field, the first value indicates the starting offset, the second value (following the colon) indicates the offset for the last byte in the field. For example, 5:7 indicates a three-byte field spanning byte offsets 5, 6, and 7.

In some cases, multi-byte fields may be variable length, in which case an alphabetic variable will be used to represent the ending offset, e.g. 5:N. Similarly, a field may following a variable length field. In this case the starting value will be shown as an offset relative to the notation used for the previous field, e.g. if the previous field were 5:N, the next field would be shown starting at N+1.

Lastly, a variable length field may follow a variable length field, in which case a relative starting offset will be shown with an alphabetic value indicating a relative ending offset, e.g. N+1:M.

(3) **Optional Fields.** When used in the *byte* column of the command tables, parentheses are used to indicate optional data byte fields. These can be absent or present at the choice of the party generating the request or response message. Devices receiving the message are required to accept any legal combination of optional data byte fields.

Unless otherwise indicated, if an optional byte field is present all prior specified byte fields must also be present. Similarly, if an optional byte field is absent all following byte fields must also be absent. For example, suppose a request accepts 4 data bytes. If data byte 3 was shown in parentheses as '(3), it would indicate that byte 3 and following were optional. A legal request could consist of just bytes [1 and 2], bytes [1, 2, and 3,] or bytes [1, 2, 3 and 4]. It is to eliminate just byte 3, but include byte 4. I.e. a request with data bytes [1, 2, and 4], would be illegal.

Multi-byte fields that are shown as optional cannot be split. Either all bytes for the field are present or absent. I.e. if a four byte multi-byte field is listed as optional, it is illegal to include the first two bytes, but not the second two bytes.

17. IPM Device “Global” Commands

This section presents the commands that are common to all Intelligent Platform Management (IPM) devices that follow this specification’s message/command interface. *This includes management controllers that connect to the system via a compatible message interface,* as well as ‘IPMB Devices’.

IPMI Management Controllers shall recognize and respond to these commands via LUN 0. Refer to *Appendix G - Command Assignments*

for the specification of the Network Function and Command (CMD) values and privilege levels for these commands. O/M = Optional/Mandatory.

Table 17-1, IPM Device ‘Global’ Commands

Command	Section	O/M
Get Device ID	17.1	M
Cold Reset	17.2	O[1]
Warm Reset	17.3	O
Get Self Test Results	17.4	M
Manufacturing Test On	17.5	O
Set ACPI Power State	17.6	O
Get ACPI Power State	17.7	O[3]
Get Device GUID	17.8	O
Broadcast Commands		
Broadcast ‘Get Device ID’	17.9	M[2]

[1] This command is not required to return a response in all implementations.

[2] Broadcast is over IPMB channels only. Request is formatted as an entire IPMB application request message, from the RsSA field through the second checksum, with the message prefixed with the broadcast slave address, 00h. Response format is same as the regular ‘Get Device ID’ response.

[3] Mandatory if *Set ACPI Power State* command is implemented on given management controller.

17.1 Get Device ID Command

This command is used to retrieve the Intelligent Device's Hardware Revision, Firmware/Software Revision, and Sensor and Event Interface Command specification revision information. The command also returns information regarding the additional 'logical device' functionality (beyond 'Application' and 'IPM' device functionality) that is provided within the intelligent device, if any.

While broad dependence on OEM-specific functionality is discouraged, two fields in the response allow software to identify controllers for the purpose of recognizing controller specific functionality. These are the Device ID and the Product ID fields. A controller that just implements standard IPMI commands can set these fields to 'unspecified'.

Table 17-2, Get Device ID Command

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

(13:16)	Auxiliary Firmware Revision Information. This field is optional. If present, it holds additional information about the firmware revision, such as boot block or internal data structure version numbers. The meanings of the numbers are specific to the vendor identified by Manufacturer ID (see below). When the vendor-specific definition is not known, generic utilities should display each byte as 2-digit hexadecimal numbers, with byte 13 displayed first as the most-significant byte.

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

Device ID/Device Instance This number is specified by the manufacturer identified by the Manufacturer ID field. The Device ID field allows controller-specific software to identify the unique application command, OEM fields, and functionality that are provided by the controller.

Controllers that have different application commands, or different definitions of OEM fields, are expected to have different Device ID values. Controllers that implement identical sets of applications commands can have the same Device ID in a given system. Thus, a 'standardized' controller could be produced where multiple instances of the controller are used in a system, and all have the same Device ID value. [The controllers would still be differentiable by their address, location, and associated information for the controllers in the Sensor Data Records.]

The Device ID is typically used in combination with the Product ID field such that the Device IDs for different controllers are unique under a given Product ID. A controller can optionally use the Device ID as an 'instance' identifier if more than one controller of that kind is used in the system. Though implementing a Device GUID is the preferred method for uniquely identifying controllers. (See *section 17.8, Get Device GUID*) *This field is binary encoded.*

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

Firmware Revision 1 Major Revision. 7-bits. This field holds the major revision of the firmware. This field shall be incremented on major changes or extensions of the functionality of the firmware - such as additions, deletions, or modifications of the command set. *This field is binary encoded and unsigned.*

The Device Available bit is used to indicate whether normal command set operation is available from the device, or it is operating in a state where only a subset of the normal commands are available. This will typically be because the device is in a firmware update state. It may also indicate that full command functionality is not available because the device is in its initialization phase or an SDR update is in progress.

Note that the revision information obtained when the Device Available bit is '1' shall be indicative of the code version that is *in effect.* Thus, the version information may vary with the Device Available bit state.

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

IPMI Version

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

*The value shall be **51h** indicating conformance with this specification, version 1.5.*

Additional Device Support

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

Manufacturer ID

This field uses the Internet Assigned Numbers Authority (http://www.iana.org/) SMI Network Management Private Enterprise Codes a.k.a. "Enterprise Numbers" for identifying the manufacturer responsible for the specification of functionality of the vendor (OEM) -specific commands, codes, and interfaces used in the controller.

For example, an event in the SEL could have OEM values in the event record. An application that parses the SEL could extract the controller address from the event record contents and use it to send the 'Get Device ID' command and retrieve the Manufacturer ID. A manufacturer-specific application could then do further interpretation based on a-priori knowledge of the OEM field, while a generic cross-platform application would typically just use the ID to present the manufacturer's name alongside uninterpreted OEM event values.

The manufacturer that defines the functionality is not necessarily the manufacturer that created the physical microcontroller. For example, Vendor A may create the controller, but it gets loaded with Vendor B's firmware. The Manufacturer ID would be for Vendor B, since they're the party that defined the controller's functionality.

The Manufacturer ID value from the *Get Device ID* command does not override Manufacturer or OEM ID fields that are explicitly defined as part of a command or record format.

If no vendor-specific functionality is defined, it is recommended that the field can either be loaded with the Manufacturer ID of the party that is responsible for the firmware for the controller, or the value FFFFh to indicate 'unspecified'. *This field is binary encoded, and unsigned.*

Product ID

This value can be used in combination with the Manufacturer ID and Device ID values to identify the product-specific element of the controller-specific functionality. This number is specified by the manufacturer identified by the Manufacturer ID field.

Typically, a controller-specific application would use the Product ID to identify the type of board, module, or system that the controller is used in, instead of using the data from the FRU information associated with the controller.

Auxiliary Firmware Revision Information

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

17.2 Cold Reset Command

This command directs the Responder to perform a 'Cold Reset' of itself. This causes default setting of interrupt enables, event message generation, sensor scanning, threshold values, and other 'power up default' state to be restored. That is, the device reinitializes its event, communication, and sensor functions. If the device incorporates a Self Test, the Self Test will also run at this time.

Table 17-3, Cold Reset Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code

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

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

17.3 Warm Reset Command

This command directs the Responder to perform a 'Warm Reset' of itself. Communications interfaces shall be reset, but current configurations of interrupt enables, thresholds, etc. will be left alone. A warm reset does not initiate the Self Test. The intent of the Warm Reset command is to provide a mechanism for cleaning up the internal state of the device and its communication interfaces. A Warm Reset will reset communication state information such as sequence number and retry tracking, but shall not reset interface configuration information such as addresses, enables, etc. An application may try a Warm Reset if it determines a non-responsive communication interface - but it must also be capable of handling the side effects.

Table 17-4, Warm Reset Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code

17.4 Get Self Test Results Command

This command directs the device to return its Self Test results, if any. A device implementing a Self Test will normally run that test on device power up as well as after Cold Reset commands. A device is allowed to update this field during operation if it has tests that run while the device is operating. Devices that do not implement a self test shall always return a 56h for this command.

While the Self Test only runs at particular times, the Get Self Test Results command can be issued any time the device is in a 'ready for commands' state.

Table 17-5, Get Self Test Results Command

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

17.5 Manufacturing Test On Command

If the device supports a 'manufacturing test mode', this command is reserved to turn that mode on. The specification of the functionality of this command is device dependent. A Cold Reset command shall, if accepted, take the device out of 'manufacturing test mode' - as shall a physical reset of the device. Device-specific commands to exit manufacturing test mode are also allowed.

Note that it may be possible to 'lock out' the command interface while in manufacturing test mode, in which case the Cold Reset command or other mechanism for exiting manufacturing test mode may fail and a physical reset of the device will be necessary to restore the device to normal operation.

The request parameters for this command are device specific. Typically, the parameters will be used for transmitting a password or key that prevents manufacturing test mode from being entered unless the correct values are provided.

Table 17-6, Manufacturing Test On

	byte	data field
Request Data	1:N	device specific parameters. See text.
Response Data	1	Completion Code

17.6 Set ACPI Power State Command

This command is provided to allow system software to tell a controller the present ACPI power state of the system. The *Set ACPI Power State command* can also be used as a mechanism for setting elements of the platform management subsystem to a particular power state. *This is an independent setting that may not necessarily match the actual power state of the system.* This command is used to enable the *reporting* of the power state, *it does not control or change the power state*.

There is corresponding information in sensor data record for the controller that tells system software which controllers require this notification. The BMC *does not* automatically inform controllers of changes in the system power state.

Since system management software does not run when the system is in a sleep state, the impact of sleep state on the platform management subsystem is mainly one of changes in the automatic handling of sensor scanning and events. For example, the system may shut down fans when in a particular power state. If the fans were monitored, shutting down the fans without notifying the platform management subsystem could cause a false failure event to be generated. Here are two possible ways to handle this:

a. Have the management controller perform the fan shut down operation after receiving the Set ACPI Power State command. In this case, the controller needs an SDR entry indicating that the controller needs to receive notification via the *Set ACPI Power State command*.

b. Have the controller monitor the system power state by proprietary means, such as a signal line directly from the power control hardware to the management controller. The management controller uses the signal to directly control the fans without receiving an *Set ACPI Power State command*. Note that that controller should still report the power state using the *Set ACPI Power State command*. This is to aid out-of-band applications that may directly access the controller to get sensor information.

Out-of-band applications should be prepared to find sensors or controllers that may have become disabled because of a sleep state. Ideally, all management controllers should remain enabled while the system is in a sleep state so that the sleep state information can be retrieved. Information in the SDR can be used to determine whether a controller gets disabled in a particular sleep state. A system will normally power up to a Legacy On state prior to the initialization of ACPI, at which time the system power state is known to be ACPI S0/G0.

Table 17-7, Set ACPI Power State Command

	byte	data field
Request Data	1	ACPI System Power State to set Power states are mutually exclusive. Only one state can be set at a time. [7] - 1b = set system power state 0b = don't change system power state [6:0] - System Power State enumeration 00h set S0 / G0 working 01h set S1 hardware context maintained, typically equates to processor/chip set clocks stopped 02h set S2 typically equates to stopped clocks with processor/cache context lost 03h set S3 typically equates to "suspend-to-RAM" 04h set S4 typically equates to "suspend-to-disk" 05h set S5 / G2 soft off 06h set S4/S5 sent when message source cannot differentiate between S4 and S5 07h set G3 mechanical off 08h sleeping sleeping - cannot differentiate between S1-S3. 09h G1 sleeping sleeping - cannot differentiate between S1-S4 0Ah set override S5 entered by override 20h set Legacy On Legacy On (indicates On for system that don't support ACPI or have ACPI capabilities disabled) 21h set Legacy Off Legacy Soft-Off 2Ah set unknown system power state unknown 7Fh no change Use this value when communicating a change the device power state without indicating a change to the system power state.
	2	ACPI Device Power State to set Power states are mutually exclusive. Only one state can be set at a time. [7] - 1 = set device power state 0 = don't change device power state [6:0] - Device Power State enumeration 00h set D0 01h set D1 02h set D2 03h set D3 2Ah set unknown 7Fh no change Use this value when communicating a change the system power state without indicating a change to the device power state.
Response Data	1	Completion Code The BMC is allowed to return an error completion code if an attempt is made to set states it knows the system doesn't support.

17.7 Get ACPI Power State Command

The command can also be used to retrieve the present power state information that has been *set into the controller*. This is an independent setting from the system power state that may not necessarily match the actual power state of the system. Unspecified bits and codes are reserved and shall be returned as 0.

Table 17-8, Get ACPI Power State Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	ACPI System Power State [7] - reserved [6:0] - System Power State enumeration 00h S0 / G0 working 01h S1 hardware context maintained, typically equates to processor/chip set clocks stopped 02h S2 typically equates to stopped clocks with processor/cache context lost 03h S3 typically equates to “suspend-to-RAM” 04h S4 typically equates to “suspend-to-disk” 05h S5 / G2 soft off 06h S4/S5 soft off, cannot differentiate between S4 and S5 07h G3 mechanical off 08h sleeping sleeping - cannot differentiate between S1-S3. 09h G1 sleeping sleeping - cannot differentiate between S1-S4 0Ah override S5 entered by override 20h Legacy On Legacy On (indicates On for system that don’t support ACPI or have ACPI capabilities disabled) 21h Legacy Off Legacy Soft-Off 2Ah unknown power state has not been initialized, or device lost track of power state.
	3	ACPI Device Power State [7] reserved [6:0] Device Power State enumeration 00h D0 01h D1 02h D2 03h D3 2Ah unknown - power state has not been initialized, or device lost track of power state.

17.8 Get Device GUID Command

This command returns a Globally Unique ID (GUID) for the management controller. The format of the ID follows that specified in the Attachment A of the *Wired for Management Baseline, Version 2.0* specification. The ID can be used to uniquely identify an instance of a management controller. This information can be used in conjunction with SDR information to verify that a particular controller is still present in the system. IPMI uses version 1 of the GUID format, with a three bit variant field of 10x (where x indicates 'don't care').

Table 17-9, Get Device GUID Command

Request Data	-	-
Response Data	1	Completion Code
	2:17	GUID bytes 1 through 16.

Note that the individual fields within the GUID are stored **least-significant byte first**, and in the order illustrated in the following table. This is the reverse of convention described in the Wired for Management Specification where GUID bytes are transmitted in 'network order' (most-significant byte first) starting with the time low field.

Table 17-10, GUID Format

GUID byte	Field	MSbyte
1	node	
2	node	
3	node	
4	node	
5	node	
6	node	MSbyte
7	clock seq and reserved	
8	clock seq and reserved	MSbyte
9	time high and version	
10	time high and version	MSbyte
11	time mid	
12	time mid	MSbyte
13	time low	
14	time low	
15	time low	
16	time low	MSbyte

17.9 Broadcast 'Get Device ID'

This is a broadcast version of the 'Get Device ID' command that is provided for the 'discovery' of Intelligent Devices on the IPMB. It is only specified for use on the IPMB. Discovery of management controllers on a PCI Management Bus is handled via the SMBus 2.0 'ARP' protocol. See [SMBUS] for more information.

To perform a 'discovery' the command is repeatedly broadcast with a different rsSA 'slave address parameter' field specified in the command. The device that has the matching physical slave address information shall respond with the same data it would return from a 'regular' (non-broadcast) 'Get Device ID' command. Since an IPMB response message carries the Responder's Slave Address, the response to the broadcast provides a positive confirmation that an Intelligent Device exists at the slave address given by the rsSA field in the request.

An application driving discovery then cycles through the possible range of IPMB Device slave addresses to find the population of intelligent devices on the IPMB. Refer to [ADDR] for information on which slave address ranges are allocated for different uses on IPMB.

Refer to the description of the *Get Device ID* command, above, for information on the fields returned by the *Broadcast Get Device ID* command response. The IPMB message format for the *Broadcast Get Device ID* request

exactly matches that for the *Get Device ID* command, with the exception that the IPMB message is prefixed with the 00h broadcast address. The following illustrates the format of the IPMB *Broadcast Get Device ID* request message:

Figure 17-1, Broadcast Get Device ID Request Message

Broadcast (00h)	rsSA	netFn/rsLUN	check1	rqSA	rqSeq/rqLUN	Cmd (01h)	check2

Addresses 00h-0Fh and F0h-FFh are reserved for I^2C functions and will not be used for IPM devices on the IPMB. These addresses can therefore be skipped if using the Broadcast Get Device ID command to scan for IPM devices. The remaining fields follow the regular IPMB definitions.

In order to speed the discovery process on the IPMB, a controller should drop off the bus as soon as it sees that the rsSA in the command doesn't match its rsSA.

18. IPMI Messaging Support Commands

This section defines the commands used to support the system messaging interfaces. This includes control bits for using the BMC as an Event Receiver and SEL Device. SMM Messaging and Event Message Buffer support is optional. Use of IPMI support for SMI's and SMM Messaging is deprecated. Configuration interface support for enabling/disabling SMM Messaging and corresponding SMI has been removed from the specification. If SMM Messaging were implemented using the IPMI infrastructure, it would now be done as an OEM-proprietary capability.

System software that is not explicitly aware of the particular platform's use of SMI Messaging must assume that the any SMI options have been pre-configured by the controller, system BIOS, or other software. Therefore, run-time system software should not reconfigure SMI options, nor should it access the Event Message Buffer if it finds that Event Message Buffer interrupt is mapped to SMI. The effects of SMS accessing the Event Message Buffer when it is configured for SMI are unspecified. Refer to *Appendix G - Command Assignments*

for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 18-1, IPMI Messaging Support Commands

Command	Section Defined	O/M
Set BMC Global Enables	18.1	M
Get BMC Global Enables	18.2	M
Clear Message Flags	18.3	M
Get Message Flags	18.4	M
Enable Message Channel Receive	18.5	O
Get Message	18.6	M[1]
Send Message	18.7	M[1]
Read Event Message Buffer	18.8	O
Get BT Interface Capabilities	18.9	M[2]
Master Write-Read	18.10	M[6]
Get System GUID	18.13	O[5]
Get Channel Authentication Capabilities	18.12	O[3]
Get Session Challenge	18.14	O[4]
Activate Session	18.15	O[4]
Set Session Privilege Level	18.16	O[4]
Close Session	18.17	O[4]
Get Session Info	18.18	O[4]
Get AuthCode	18.19	O
Set Channel Access	18.20	O[4]
Get Channel Access	18.21	O[4]
Get Channel Info Command	18.22	O[4]
Set User Access Command	18.23	O[4]
Get User Access Command	18.24	O[4]
Set User Name	18.25	O[5]
Get User Name Command	18.26	O[4]
Set User Password Command	18.27	O[4]

1. Optional if the System Interface is the only channel that's implemented.
2. Mandatory only if BT (block transfer) System Interface is used.
3. Mandatory if Private Bus FRU SEEPROMs or IPMB implemented.
4. Mandatory if session-based channels are supported
5. Highly recommended for session-based channels. It is also recommended that the implementation support multiple of users with configurable usernames.

6. Mandatory for a BMC that includes IPMB or PCI SMBus channels, or for any BMC or satellite controller that implements a private management bus for FRU SEEPROM access.

18.1 Set BMC Global Enables Command

This command is used to enable message reception into Message Buffers, and any interrupt associated with that buffer getting full. The OEM0, OEM 1, and OEM 2 flags are available for definition by the OEM/System Integrator. Generic system management software must not alter these bits.

Table 18-2, Set BMC Global Enables Command

	byte	data field
Request Data	1	This field is set to xxxx_100xb on power-up and system resets. If the implementation allows the receive message queue interrupt to be enabled/disabled, the default for bit 0 should be 0b, otherwise it should always be 1b. [7] OEM 2 Enable. Generic system mgmt. software must do a 'read-modify-write' using the *Get BMC Global Enables* and *Set BMC Global Enables* to avoid altering this bit. [6] OEM 1 Enable. Generic system mgmt. software must do a 'read-modify-write' using the *Get BMC Global Enables* and *Set BMC Global Enables* to avoid altering this bit. [5] OEM 0 Enable. Generic system mgmt. software must do a 'read-modify-write' using the *Get BMC Global Enables* and *Set BMC Global Enables* to avoid altering this bit. [4] reserved [3] 1b = Enable System Event Logging (enables/disables logging of events to the SEL - with the exception of events received over the system interface. Event reception and logging via the system interface is always enabled.) SEL Logging is enabled by default whenever the BMC is first powered up. It's recommended that this default state also be restored on system resets and power on. [2] 1b = Enable Event Message Buffer. Error completion code returned if written as '1' and Event Message Buffer not supported. [1] 1b = Enable Event Message Buffer Full Interrupt [0] 1b = Enable Receive Message Queue Interrupt (this bit also controls whether KCS communication interrupts are enabled or disabled. An implementation is allowed to have this interrupt always enabled.)
Response Data	1	Completion Code.

18.2 Get BMC Global Enables Command

This command is used to retrieve the present setting of the Global Enables. The OEM0, OEM 1, and OEM 2 flags are available for definition by the OEM/System Integrator. Generic system management software must ignore these bits.

Table 18-3, Get BMC Global Enables Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	[7] - 1b = OEM 2 Enabled. [6] - 1b = OEM 1 Enabled. [5] - 1b = OEM 0 Enabled. [4] - reserved [3] - 1b = System Event Logging Enabled [2] - 1b = Event Message Buffer Enabled [1] - 1b = Event Message Buffer Full Interrupt Enabled [0] - 1b = Receive Message Queue Interrupt Enabled (this bit also indicates whether KCS communication interrupt are enabled or disabled.)

18.3 Clear Message Flags Command

This command is used to flush unread data from the Receive Message Queue or Event Message Buffer. This will also clear the associated buffer full / message available flags. See *Get Message Flags* command.

Table 18-4, Clear Message Flags Command

	byte	data field
Request Data	1	[7] - 1b = Clear OEM 2 [6] - 1b = Clear OEM 1 [5] - 1b = Clear OEM 0 [4] - reserved [3] - 1b = Clear watchdog pre-timeout interrupt flag [2] - reserved [1] - 1b = Clear Event Message Buffer. [0] - 1b = Clear Receive Message Queue.
Response Data	1	Completion Code. Implementations are not required to return an error completion code if an attempt is made to clear the Event Message Buffer flag but the Event Message Buffer is not supported.

18.4 Get Message Flags Command

This command is used to retrieve the present ‘message available’ states. The OEM0, OEM 1, and OEM 2 flags are available for definition by the OEM/System Integrator. Generic system management software must ignore these bits.

Table 18-5, Get Message Flags Command

	byte	data field
Request Data	1	-
Response Data	1	Completion Code.
	2	Flags [7] - 1b = OEM 2 data available. [6] - 1b = OEM 1 data available. [5] - 1b = OEM 0 data available. [4] - reserved [3] - 1b = Watchdog pre-timeout interrupt occurred [2] - reserved [1] - 1b = Event Message Buffer Full. (Return as 0 if Event Message Buffer is not supported, or when the Event Message buffer is disabled.) [0] - 1b = Receive Message Available. One or more messages ready for reading from Receive Message Queue

18.5 Enable Message Channel Receive Command

This command is used to enable/disable message reception into the Receive Message Queue from a given message channel. The command provides a mechanism to allow SMS to only receive messages from channels that it intends to process, and provides a disable mechanism in case the receive message queue is being erroneously or maliciously flooded with requests on a particular channel. It does not affect the ability for SMS to transmit on that channel. *Only the SMS Message channel is enabled by default. All other channels must be explicitly enabled by BIOS or system software, as appropriate.* It is recommended that a 'Destination Unavailable' completion code be returned if a request message to SMS is rejected because reception has been disabled.

Table 18-6, Enable Message Channel Receive Command

	byte	data field
Request Data	1	Channel Number [7:4] - reserved [3:0] - channel number
	2	Channel State [7:2] - reserved [1:0] - 00b = disable channel 01b = enable channel 10b = get channel enable/disable state 11b = reserved
Response Data	1	Completion Code
	2	Channel Number [7:4] - reserved [3:0] - channel number
	3	Channel State [7:1] - reserved [0] - 1b = channel enabled 0b = channel disabled

18.6 Get Message Command

This command is used to get data from the Receive Message Queue. Refer to *Table 6-8, IPMI Message and IPMB / Private Bus Transaction Size Requirements,* for information that can be used to determine the sizes of messages that need to be supported for a given Receive Message Queue implementation.

Software is responsible for reading all messages from the message queue even if the message is not the expected response to an earlier *Send Message.* System management software is responsible for matching responses up with requests.

The *Get Message* command includes an "inferred privilege level" that is returned with the message. This can help avoid the need for software to implement a separate privilege-level and authentication mechanism. This works as follows: Suppose a user activates a session with a maximum privilege level of Administrator on a multi-session channel, and that an MD5 authentication type was negotiated. Also suppose that User-level authentication is disabled. A user that has User or higher privilege can place messages into the receive message queue by sending them to LUN 10b, or by using the *Send Message* command. If the packet has Authentication Type = MD5, the packet will be assigned an inferred privilege level based the on the present operating privilege level for the user (set using the *Set Session Privilege Level* command). If, before sending the packet, the user had set their privilege level to Operator, the packet would be assigned an inferred privilege level of Operator. (Note that this means an authenticated (signed) packet can be assigned different inferred privilege levels based on the present operating privilege set by the *Set Session Privilege Level* command.) If the message is received in a packet that has Authentication Type = None, the packet will be assigned an inferred privilege level of 'User', since that is the lowest privilege level for which that type of authentication is accepted.

Now suppose that the remote user had used the receive message queue as a way to send a message to system management software that requests a soft shutdown of the operating system. The message would either have an inferred privilege level of 'Operator' or 'User' depending on whether it was sent as an authenticated message or not. System Management Software can then use that inferred privilege level as part of deciding whether to accept and process the message or not. For single-session channels, the inferred privilege level is always set to the present operating privilege level. For session-less channels, the inferred privilege level is set to 'None', indicating that there was no IPMI-specified authentication operating on the channel from which the message was received.

Table 18-7, Get Message Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code generic, plus following command specific completion code: 80h = data not available (queue / buffer empty) Implementation of this completion code is Mandatory. The code eliminates the need for system software to always check the Message Buffer Flags to see if there data left in the Receive Message Queue. If a non-OK, non-80h completion is encountered - software will need to check the Message Flags to get the empty/non-empty status of the Receive Message Queue.
	2	Channel Number [7:4] Inferred privilege level for message. When the BMC receives a message for the receive message queue, it assigns an 'inferred privilege level' to the message as follows: If the message is received from a session-based channel, it will initially be assigned a privilege level that matches the 'maximum requested privilege level' that was negotiated via the *Activate Session* command. If per-message authentication is enabled, but User-level authentication is disabled, the BMC will assign a level of 'User' to any messages that are received with an Authentication Type = none. (Note that per-message and user-level authentication options only apply to multi-session channels) The BMC will then lower the assigned privilege limit, if necessary, based on the present session privilege limit that was set via the *Set Session Privilege Level* command. If the channel is session-less (e.g. IPMB), the BMC will return 'None' for the privilege level. 0h = None (unspecified) 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM Proprietary level [3:0] channel number
	3:N	Message Data. Max. Length & format dependent on protocol associated with channel.

The following table indicates the contents of the Message Data field from the *Get Message* response according to the Channel Type and Channel Protocol that was used to place the message in the Receive Message Queue.

Table 18-8, Get Message Data Fields

	Originating Channel Type	Channel Protocol	Message Data for received requests(RQ) and responses (RS)
1	IPMB (I^2C)	IPMB	RQ: netFn/rsLUN, chk1, rqSA, rqSeq/rqLUN, cmd, <data>, chk2
			RS: netFn/rqLUN, chk1, rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
2	ICMB v1.0	ICMB-1.0	See *Section 8.2, ICMB Bridge Commands in BMC using Channels*
3	ICMB v0.9	ICMB-0.9	See *Section 8.2, ICMB Bridge Commands in BMC using Channels*
4	802.3 LAN	IPMB	RQ: Session Handle, rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2
			RS: Session Handle, rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
5	Asynch. Serial/Modem (RS-232)	IPMB (Basic Mode, Terminal Mode, and PPP Mode)	RQ/RS: See row for Originating Channel Type = 802.3 LAN Note: When LUN 10b is used to deliver a message to SMS from a Terminal Mode remote console, the BMC inserts fixed values for the SWIDs and LUNs in the message. Messages from the remote console are always returned as coming from SWID 40h (81h) LUN 00b, and going to SMS SWID 20h (41h) LUN 00b.
6	Other LAN	IPMB	See row for Originating Channel Type = 802.3 LAN
7	PCI SMBus	IPMI-SMBus	RQ: rsSA, Netfn(even)/rsLUN, 00h, rqSA, rqSeq/rqLUN, CMD, <data>, PEC RS: rqSA or rqSWID, NetFn(odd)/rqLUN, 00h, rsSA or rsSWID, rqSeq/rsLUN, CMD, completion code, <data>, PEC
8	SMBus v1.0/1.1		
9	SMBus v2.0		
10	reserved for USB 1.x	n/a	n/a
11	reserved for USB 2.x	n/a	n/a
12	System Interface	BT, KCS, SMIC	RQ/RS: See row for Originating Channel Type = 802.3 LAN.

18.7 Send Message Command

The Send Message command is used for bridging IPMI messages between channels, and between the system management software (SMS) and a given channel. Refer to *6.12, BMC Message Bridging*, for information on how the *Send Message* command is used.

Table 18-9, Send Message Command

	byte	data field
Request Data	1	Channel Number [7:6] 00b = No tracking. The BMC reformats the message for the selected channel but does not track the originating channel, sequence number, or address information. This option is typically used when software sends a message from the system interface to another media. Software will typically use 'no tracking' when it delivers sends a message from the system interface to another channel, such as IPMB. In this case, software will format the encapsulated message so that when it appears on the other channel, it will appear to have been directly originated by BMC LUN 10b. See *6.12.1, BMC LUN 10b Routing*. 01b = Track Request. The BMC records the originating channel, sequence number, and addressing information for the requester, and then reformats the message for the protocol of the destination channel. When a response is returned, the BMC looks up the requester's information and format the response message with the framing and destination address information and reformats the response for delivery back to the requester. This option is used for delivering IPMI Request messages from non-SMS (non-system interface) channels. See *6.12.3, Send Message Command with Response Tracking*. 10b = Send Raw. (optional) This option is primarily provided for test purposes. It may also be used for proprietary messaging purposes. The BMC simply delivers the encapsulated data to the selected channel in place of the IPMI Message data. Note that if the channel uses sessions, the first byte of the Message Data field must be a Session Handle. The BMC should return a non-zero completion code if an attempt is made to use this option for a given channel and the option is not supported. It is recommended that completion code CCh be returned for this condition. 11b = reserved [5:4] reserved [3:0] channel number to send message to.
	2:N	Message Data. Format dependent on target channel type. See *Table 18-10, Message Data for Send Message Command*
Response Data	1	Completion Code generic, plus additional command-specific completion codes: 80h = Invalid Session Handle. The session handle does not match up with any currently active sessions for this channel. <u>If channel medium = IPMB, SMBus, or PCI Management Bus:</u> (This status is important for applications that need to access low-level I^2C or SMBus devices and should be implemented.) 81h = Lost Arbitration 82h = Bus Error 83h = NAK on Write
	(2)	Response Data This data will only be present when using the *Send Message* command to originate requests from IPMB or PCI Management Bus to other channels such as LAN or serial/modem. It is not present in the response to a *Send Message* command delivered via the System Interface.

NOTE: The BMC does not parse messages that are encapsulated in a Send Message command. Therefore, it does not know what privilege level should associated with an encapsulated message. Thus, messages that are sent to a session using the *Send Message* command are always output using the Authentication Type that was negotiated when the session was activated.

The following table summarizes the contents of the Message Data field when the *Send Message* command is used to deliver an IPMI Message to different channel types. Note that in most cases the format of message information the Message Data field follows that used for the IPMB, with two typical exceptions: When the message is delivered to channels without physical slave devices, a software ID (SWID) field takes the place of the slave address field. When the message is delivered to a channel that supports sessions, the first byte of the message data holds a Session Handle.

Table 18-10, Message Data for Send Message Command

	Target Channel Type	Target Channel Protocol	Message Data field contents for *Send Message* command for sending requests(RQ) and responses (RS)
1	IPMB (I^2C)	IPMB	RQ: rsSA, netFn/rsLUN, chk1, rqSA, rqSeq/rqLUN, cmd, <data>, chk2
			RS: rqSA, netFn/rqLUN, chk1, rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
2	ICMB v1.0	ICMB-1.0	See *Section 8.2, ICMB Bridge Commands in BMC using Channels*
3	ICMB v0.9	ICMB-0.9	See *Section 8.2, ICMB Bridge Commands in BMC using Channels*
4	802.3 LAN	IPMB+session header	RQ: Session Handle[1], rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2
			RS: Session Handle[1], rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
5	Asynch. Serial/Modem (RS-232)	IPMB (Basic Mode, Terminal Mode, and PPP Mode)	RQ/RS: See Target Channel Type = 802.3 LAN Note: Terminal mode has a single, fixed SWID for the remote console, software using Send Message to deliver a message to a terminal mode remote console should use their SWID or slave address as the source of the request or response, and the Terminal Mode SWID (40h) as the destination.
6	Other LAN	IPMB	See Target Channel Type = 802.3 LAN
7	PCI SMBus	IPMI-SMBus	See Target Channel Type = IPMB
8	SMBus v1.0/1.1		
9	SMBus v2.0		
10	reserved for USB 1.x	n/a	n/a
11	reserved for USB 2.x	n/a	n/a
12	System Interface		RQ/RS: See Target Channel Type = IPMB Note: BMC adds Session Handle info when it puts the message into the Receive Message Queue.

1. Session Handle. Identifies the particular active session for this channel. The session handle identifies a particular active session on the given channel. The BMC assigns a different value to each time a new session is activated. A typical implementation will keep track of the last value that was assigned and increment it before assigning it to a new active session when the *Activate Session* command has been accepted. Software must include this field for channels where the *Get Channel Info* command indicates that the channel supports sessions.

18.8 Read Event Message Buffer Command

This command is used to retrieve the contents of the Event Message Buffer. Reading the buffer automatically clears the Event Message Buffer Full flag from the *Get Message Flags* command.

Table 18-11, Read Event Message Buffer Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code. generic, plus additional command-specific completion codes: 80h = data not available (queue / buffer empty)
	2:17	Message Data. 16 bytes of data in SEL Record format, per *Table 26-1, SEL Event Records*. A dummy Record ID of FFFFh should be returned for events placed in the Event Message Buffer while event logging is disabled or if the SEL is full. System management software should ignore the record ID when event logging is disabled.

18.9 Get BT Interface Capabilities Command

The BT interface includes a *Get BT Interface Capabilities* command that returns various characteristics of the interface, including buffer sizes, and multithreaded communications capabilities.

Table 18-12, Get BT Interface Capabilities Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Number of outstanding requests supported (1 based. 0 illegal)
	3	Input (request) buffer size in bytes. (1 based.)
	4	Output (response) buffer size in bytes. (1 based.)
	5	BMC Request-to-Response time, in seconds, 1 based. 30 seconds, maximum.
	6	Recommended retries (1 based). (see text for BT Interface)

18.10 Master Write-Read Command

This command can be used for low-level I^2C/SMBus write, read, or write-read accesses to the IPMB or private busses behind a management controller. The command can also be used for providing low-level access to devices that provide an SMBus slave interface.

Table 18-13, Master Write-Read Command

	byte	data field
Request Data	1	bus ID: [7:4] channel number [3:1] bus ID, 0-based (always 000b for public bus) [0] bus type: 0 = public (e.g. IPMB or PCI Management Bus) 1 = private bus
	2	[7:1] - Slave Address [0] - reserved. Write as 0.
	3	Read count. Number of bytes to read, 1 based. 0 = no bytes to read. *The maximum read count should be at least 34 bytes.* This allows the command to be used for an SMBus Block Read. This is required if the command provides access to an SMBus or IPMB. Otherwise, if FRU SEEPROM devices are accessible, at least 31 bytes must be supported. Note that an implementation can support fewer bytes can be supported if none of the devices to be accessed can handle the recommended minimum.
	4:N	Data to write. This command should support *at least 35 bytes* of write data. This allows the command to be used for an SMBus Block Write with PEC. Otherwise, if FRU SEEPROM devices are accessible, at least 31 bytes must be supported. Note that an implementation is allowed to support fewer bytes if none of the devices to can handle the recommended minimum.
Response Data	1	Completion Code A management controller shall return an error Completion Code if an attempt is made to access an unsupported bus. generic, plus following command specific codes: 81h = Lost Arbitration 82h = Bus Error 83h = NAK on Write 84h = Truncated Read
	(2:M)	Bytes read from specified slave address. This field will be absent if the read count is 0. The controller terminates the I^2C transaction with a STOP condition after reading the requested number of bytes.

18.11 Session Header Fields

Whether the session header fields are present in a packet is based on whether the channel is specified as supporting multiple sessions or not. In addition, which session fields are present is based on the authentication type. Single-session connections and session-less channels do not include session header fields.

Session header fields are present on *all* packets where the channel and connection mode is specified as supporting multiple sessions, even if the particular implementation only supports one session. The following tables for the *Get System GUID, Get Channel Authentication Capabilities, Get Session Challenge,* and *Activate Session* commands explicitly show 'session header' fields in gray. This is done because those commands can be executed prior to a session being activated, and therefore certain session header field values are handled differently than they are after a session is established.

The grayed session header fields illustrate which session header fields are present and what their values are required to be, but they do not serve to specify the order or organization of those fields in the packet for a particular medium. Refer to the example packet format figures for (e.g. *Table 12-8, RMCP Packet for IPMI via*

Ethernet) for the specification of the organization and position of the session header bytes for a particular medium.

The following applies to packets on connections that are specified with support for multiple sessions:

- Session header fields are present on *all* packets where the channel and connection mode is specified as supporting multiple sessions, even if the particular implementation only supports one session.
- Note that the command tables do not show the session header fields except for the *Get Channel Authentication Capabilities*, *Get Session Challenge*, and *Activate Session* commands. However, they are still required for all commands on a multi-session connection.
- The Authentication Code field in the session header may or may not be present based on the Authentication Type. The authentication code field is absent whenever the Authentication Type is NONE. Whether the authentication code field is present or not when the Authentication Type = OEM is dependent on the OEM identified in the *Get Channel Authentication Capabilities* command.
- If per-message authentication is turned off for the channel, only the *Activate Session* command would use a non-NONE authentication type and include an AuthCode signature. All other commands under the session are sent with Authentication Type = NONE.
- If per-message authentication is turned off and a packet is received that has a non-NONE authentication type, it will be accepted (if the authentication type is supported), however the implementation is not required to authenticate the packet. Note that an implementation *may* authenticate the packet, therefore the Authentication Code must be correct.
- If User authentication is turned off for the channel, the behavior is the same as if per-message authentication is turned off, except that only packets for commands that are enabled at User privilege level are sent with Authentication Type = NONE. All other packets must be authenticated per the Authentication Type used to activate the session.

18.12 Get Channel Authentication Capabilities Command

This command is sent in unauthenticated (clear) format. This command is used to retrieve capability information about the channel that the message is delivered over, or for a particular channel. The command returns the authentication algorithm support for the given privilege level. When activating a session, the privilege level passed in this command will normally be the same Requested Maximum Privilege level that will be used for a subsequent *Activate Session* command.

The *Get Channel Authentication Capabilities* command can also be used as a no-op ‘Ping’ to keep a session from timing out.

Table 18-14, Get Channel Authentication Capabilities Command

Session Request Data		authentication type = NONE
		session seq# = null (0's)
		session ID = null (0's)
		AuthCode = NOT PRESENT
IPMI Request Data	1	Channel Number [7:4] - reserved [3:0] - channel number. 0h-7h, Fh = channel numbers Eh = retrieve information for channel this request was issued on.
	2	Requested Maximum Privilege Level [7:4] - reserved [3:0] - requested privilege level 0h = reserved 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM Proprietary level

Session Response Data		authentication type = NONE
		session seq# = null (0's)
		session ID = null (0's)
		AuthCode = NOT PRESENT
IPMI Response Data	1	Completion Code
	2	Channel Number Channel number that the Authentication Capabilities is being returned for. If the channel number in the request was set to Eh, this will return the channel number for the channel that the request was received on.
	3	Authentication Type Support Returns the setting of the Authentication Type Enable field from the configuration parameters for the given channel that corresponds to the Requested Maximum Privilege Level. [7:6] - reserved [5:0] - Authentication type(s) enabled for given Requested Maximum Privilege Level All bits: 1b = supported 0b = authentication type not available for use. [5] - OEM proprietary (per OEM identified by the IANA OEM ID in the RMCP Ping Response) [4] - straight password / key [3] - reserved [2] - MD5 [1] - MD2 [0] - none
	4	[7:5] - reserved [4] - Per-message Authentication status 0b = Per-message Authentication is enabled. Packets to the BMC must be authenticated per Authentication Type used to activate the session, and User Level Authentication setting, following. 1b = Per-message Authentication is disabled. Authentication Type 'none' accepted for packets to the BMC after the session has been activated. [3] - User Level Authentication status 0b = User Level Authentication is enabled. User Level commands must be authenticated per Authentication Type used to activate the session. 1b = User Level Authentication is disabled. Authentication Type 'none' accepted for User Level commands to the BMC. [2:0] - Anonymous Login status (see sections *6.9.1, 'Anonymous Login' Convention* and *6.9.2, Anonymous Login*) [2] - 1b = Non-null user names enabled [1] - 1b = Null user name enabled (A user that has a null username, but non-null password, is presently enabled) [0] - 1b = Anonymous Login enabled (A user that has a null username and null password is presently enabled)
	5	reserved
	6:8	OEM ID. IANA Enterprise Number for OEM/Organization that specified the particular OEM Authentication Type. Least significant byte first. Return 00h, 00h, 00h if no OEM authentication type available.
	9	OEM auxiliary data. Additional OEM-specific information for the OEM Authentication Type. Return 00h if no OEM authentication type available.

18.13 Get System GUID Command

This optional, though highly recommended, command can be used to return a globally unique ID (GUID) for the system to support the remote discovery process and other operations. Since the GUID is typically 'permanently' assigned to a system, an interface that would allow the GUID to be configured or changed is not specified. The session header (Session Request data and Session Response Data) values shown illustrate the values that would be used to execute a *Get System GUID* Command outside of an active session. The Get System GUID will always be accepted outside of an active session.

The *Get System GUID* command can also be executed in the context of an active session (providing the user is operating at higher than 'Callback' privilege). When the *Get System GUID* command is executed in the context of an active session, the session header fields must have correct values according to the authentication, session ID, and session sequence number information that was negotiated for the session.

Table 18-15, Get System GUID Command

Session Request Data		authentication type = NONE
		session seq# = null (0's)
		session ID = null (0's)
		AuthCode = NOT PRESENT
Request Data	-	-
Session Response Data		authentication type = NONE
		session seq# = null (0's)
		session ID = null (0's)
		AuthCode = NOT PRESENT
Response Data	1	Completion Code
	2:17	GUID bytes 1 through 16. See *Table 17-10, GUID Format*.

18.14 Get Session Challenge Command

This command is sent in unauthenticated format. While a session ID is returned from the response to the *Get Session Challenge* command, the session must be activated using the *Activate Session* command before it can be used for sending other authenticated commands. The *Activate Session* command provides the starting sequence number for subsequent messages under the session.

When the management controller looks up user names the controller scans the names sequentially by user ID starting from User ID 1. Disabled user names are skipped. The scan stops when the first matching enabled user name is made.

Table 18-16, Get Session Challenge Command

	byte	data field
Session Request Data		authentication type = NONE
		session seq# = null (0's)
		session ID = null (0's)
		AuthCode = NOT PRESENT
IPMI Request Data	1	Authentication Type for Challenge [7:4] - reserved [3:0] - requested Authentication Type 0h = none. No hashing or authentication done on session packets. Authentication Code field is not present. 1h = MD2 2h = MD5 3h = reserved 4h = straight password / key 5h = OEM proprietary all other = reserved
	2:17	User Name. Sixteen-bytes. All 0's for null user name (User 1)
Session Response Data		authentication type = NONE
		session seq# = null (0's)
		session ID = null (0's)
		AuthCode = NOT PRESENT
IPMI Response Data	1	Completion Code 81h = invalid user name 82h = null user name (User 1) not enabled
	2:5	Temporary Session ID. LS byte first. This is a provision for a temporary Session ID that can be given out to parties that have requested challenges, but have not yet activated a session. It can be used as a mechanism to help protect against denial of service attacks by grabbing all free session IDs.
	6:21	Challenge string data

18.15 Activate Session Command

While a Session ID is returned from the response to the *Get Session Challenge* command, the session must be activated using the *Activate Session* command before it can be used for sending other authenticated commands.

The initial *Activate Session* command is used by the remote console to set the starting sequence number for subsequent messages under the session. When the *Activate Session* command is issued (for a given session ID) the outbound session sequence number is set by the remote console and can be any random value.

For a given temporary session ID, the BMC must accept *Activate Session* commands with a null session sequence number and silently discard all other commands targeted to that session ID. This provision is to enable a remote console to retry the *Activate Session* command in case the response was lost. The BMC will continue to accept the Activate Session command with a null session sequence number until the first valid and appropriately authenticated command with a non-null session sequence number is received. (The non-null sequence number must also be within the range specified by the initial inbound sequence number). After which, all subsequent commands for the session must have appropriately incremented, non-null sequence number values, including any *Activate Session* commands that may be received during session operation.

The remote console can use an *Activate Session* command to change the outbound session sequence number during session operation. The BMC may also elect to change its inbound session sequence number at that time, or may continue with the inbound session sequence number sequence already in progress.

Table 18-17, Activate Session Command

	byte	data field
Session Request Data		authentication type = from corresponding *Get Session Challenge* command.
		session seq# = null (0's) when in 'pre-session' phase, non-null afterward. See text.
		session ID = Temporary Session ID value from corresponding *Get Session Challenge* command, or present session ID if session already active
		AuthCode = present unless authentication type = None. See *18.15.1, AuthCode Algorithms* for information on calculating this field for authentication types that are not "None".
IPMI Request Data	1	Authentication Type for session. The selected type will be used for session activation and for all subsequent authenticated packets under the session, unless 'Per-message Authentication' or 'User Level Authentication' are disabled. (See *6.11.4, Per-Message and User Level Authentication* Disables, for more information.) [7:4] - reserved [3:0] - Authentication Type. This value must match with the Authentication Type used in the *Get Session Challenge* request for the session. In addition, for multi-session channels this value must also match the authentication type used in the Session Header. 0h = none. No hashing or authentication done on session packets. Authentication Code field is not present. 1h = MD2 2h = MD5 3h = reserved 4h = straight password / key 5h = OEM proprietary all other = reserved
	2	Maximum privilege level requested. Indicates the highest privilege level that may be requested for this session. This privilege level must be less than or equal to the privilege limit for the channel and the privilege limit for the user in order for the *Activate Session* command to be successful (completion code = 00h). Once the *Activate Session* command has been successful, the requested privilege level becomes a 'session limit' that cannot be raised beyond the requested level, even if the user and/or channel privilege level limits would allow it. I.e. it takes precedence over the channel and user privilege level limits. [7:4] - reserved [3:0] - Requested Maximum Privilege Level 0h = reserved 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM Proprietary level all other = reserved
	3:18	For multi-session channels: (e.g. LAN channel): Challenge String data from corresponding *Get Session Challenge* response. For single-session channels that lack session header (e.g. serial/modem in Basic Mode): Clear text password or AuthCode. See *18.15.1, AuthCode Algorithms*.
	19:22	Initial Outbound Sequence Number = Starting sequence number that remote console wants used for messages from the BMC. (LS byte first). Must be non-null in order to establish a session. 0000_0000h = reserved. If the Activate Session command is executed after a session has been established, the Outbound Sequence Number will be reset to the given value. This will take effect for the corresponding Activate Session response and subsequent commands under the session.

Session Response Data		session ID = value from request
		authentication type = value passed in from request data
		session seq# = Initial outbound sequence number from corresponding Activate Session request.
		AuthCode = present unless authentication type = None. See *18.15.1, AuthCode Algorithms* for information on calculating this field for authentication types that are not “None”.
IPMI Response Data	1	Completion Code 00h = success 81h = No session slot available (BMC cannot accept any more sessions) 82h = No slot available for given user. (Limit of user sessions allowed under that name has been reached) 83h = No slot available to support user due to maximum privilege capability. (An implementation may only be able to support a certain number of sessions based on what authentication resources are required. For example, if User Level Authentication is disabled, an implementation may be able to allow a larger number of users that are limited to User Level privilege, than users that require higher privilege.) 84h = session sequence number out-of-range 85h = invalid session ID in request 86h = requested maximum privilege level exceeds user and/or channel privilege limit
	2	Authentication Type for remainder of session The primary use of this parameter is to report whether per-message authentication will be used for IPMI message packets that follow the *Activate Session* packet. Per-message authentication is a channel configuration option that is set using the *Get User Name* command. If per-message authentication is disabled, the Authentication Type will be returned as ‘none’, and all subsequent packets for the session are required to use ‘none’ as the authentication type. Otherwise this value will be set to the Authentication Type that was used in the request. Note that *Activate Session* requests and responses are always required to be authenticated per what is returned by the *Get Session Challenge* command for the user. [7:4] - reserved [3:0] - Authentication Type 0h = none. No hashing or authentication done on session packets. Authentication Code field is not present. 1h = MD2 2h = MD5 3h = reserved 4h = straight password / key 5h = OEM proprietary all other = reserved
	3:6	Session ID - use this for remainder of session. While atypical, the BMC is allowed to change the session ID from the one that passed in the request.
	7:10	Initial inbound seq# = Sequence number that BMC wants remote console to use for subsequent messages in the session. The BMC returns a non-null value for multi-session connections and returns null (all 0’s) for single-session connections.
	11	Maximum privilege level allowed for this session [7:4] - reserved [3:0] - Maximum Privilege Level allowed 0h = reserved 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM Proprietary level all other = reserved

18.15.1 AuthCode Algorithms

The following lists the AuthCode calculation mechanism and field usage. The *Get AuthCode* command uses the same algorithm as

- Refer to the [RFC1319] and [RFC1321] for information on the MD2 and MD5 algorithms, respectively.
- For the following table, '+' indicates concatenation of data, and H( ) represents the application of the message digest algorithm to that data.
- The data bytes are passed to the message-digest algorithm in the same order that they're transmitted in the message / packet.
- The password/key is 0 padded to 16-bytes for all specified authentication types.

Figure 18-1, AuthCode Algorithms

Authentication Type	Algorithm
	Single Session AuthCode carried in IPMI message data for *Activate Session Command*
straight password	AuthCode = password
MD2	AuthCode = H(password + temporary session ID + challenge string+ password)
MD5	AuthCode = H(password + temporary session ID + challenge string+ password)
	Multi-Session AuthCode carried in session header for all 'authenticated' packets
straight password	AuthCode=password
MD2	AuthCode = H(password + session ID[1] + *IPMI Message data* + session_seq# + password)
MD5	AuthCode = H(password + session ID[1] + *IPMI Message data* + session_seq# + password)
	Get AuthCode AuthCode carried in IPMI message data, per command description
straight password	See description of *Get AuthCode* command.
MD2	AuthCode = H(password + Get AuthCode data + password)
MD5	AuthCode = H(password + Get AuthCode data + password)

1. This will be the Temporary Session ID when calculating the AuthCode for the initial *Activate Session* command.

18.16 Set Session Privilege Level Command

This command is sent in authenticated format. When a session is activated, the session is set to an initial privilege level. A session that is activated at a maximum privilege level of Callback is set to an initial privilege level of Callback and cannot be changed. All other sessions are initially set to USER level, regardless of the maximum privilege level requested in the *Activate Session* command. The remote console must 'raise' the privilege level of the session using this command in order to execute commands that require a greater-than-User level of privilege.

This command cannot be used to set a privilege level higher than the lowest of the privilege level set for the user (via the *Set User Access* command) and the privilege limit for the channel that was set via the *Set Channel Access* command. Note that the specification allows a session to be used across multiple channels. The maximum privilege limit and authentication are based on the user privilege and channel limits. Since these can vary on a per channel basis, an implementation cannot simply assign a single privilege limit to a given session but must authenticate incoming messages according to the specific settings for the channel and the user on a per-channel basis.

Table 18-18, Set Session Privilege Level Command

IPMI Request Data	1	Requested Privilege Level [7:4] - reserved [3:0] - Privilege Level 0h - no change, just return present privilege level 1h - reserved 2h - change to USER level 3h - change to OPERATOR level 4h - change to ADMINISTRATOR level 5h - change to OEM Proprietary level all other = reserved
IPMI Response Data	1	Completion Code. Generic, plus following command specific: 80h = Requested level not available for this user 81h = Requested level exceeds Channel and/or User Privilege Limit 82h = Cannot disable User Level authentication
	2	New Privilege Level (or present level if 'return present privilege level' was selected.)

18.17 Close Session Command

This command is used to immediately terminate a session in progress. It is typically used to close the session that the user is communicating over, though it can be used to other terminate sessions in progress (provided that the user is operating at the appropriate privilege level, or the command is executed over a local channel - e.g. the system interface).

Table 18-19, Close Session Command

	byte	data field
Request Data	1:4	Session ID
Response Data	1	Completion Code 87h = invalid session ID in request

18.18 Get Session Info Command

This command is used to get information regarding which users presently have active sessions, and, when available, addressing information for the party that has established the session. Note that a portion of the response is dependent on the type of channel.

Table 18-20, Get Session Info Command

IPMI Request Data	1	Session Index: This value is used to select entries in a logical 'sessions' table maintained by the management controller. Info for all active sessions can be retrieved by incrementing the session index from 1 to N, where N is the number of entries in the Active Sessions table. 00h = Return info for active session associated with session this command was received over. N = get info for Nth active session FEh = Look up session info according to Session Handle passed in this request. FFh = Look up session info according to Session ID passed in this request.
		Present if Session Index = FEh:
	2	Session Handle. 00h = reserved.
		Present if Session Index = FFh:
	2:5	Session ID. ID of session to look up session information for.
IPMI Response Data	1	Completion Code
	2	Session Handle presently assigned to active session. FFh = reserved.
	3	Number of possible active sessions. This value reflects the number of possible entries (slots) in the sessions table. [7:6] - reserved [5:0] - session slot count. 1-based.
	4	Number of currently active sessions on all channels on this controller. [7:6] - reserved [5:0] - active session count. 1-based. 0 = no currently active sessions.
		The following parameters are returned only if there is an active session corresponding to the given session index:
	5	User ID for selected active session [7:6] - reserved. [5:0] - User ID. 000000b = reserved.
	6	Operating Privilege Level [7:4] - reserved [3:0] - present privilege level that user is operating at.
	7	Channel that session was activated over. [7:4] - reserved [3:0] - channel number
		The following bytes 8:18 are returned if Channel Type = 802.3 LAN:
	8:11	IP Address of remote console (MS-byte first). Address that was received in the *Activate Session* command that activated the session.
	12:17	MAC Address (MS-byte first). Address that was received in the *Activate Session* command that activated the session.
	18:19	Port Number of remote console (LS-byte first). Port Number that was received in UPD packet that held the *Activate Session* command that activated the session.
		The following bytes 8:13 are returned if Channel Type = asynch. serial/modem:
	8	Session / Channel Activity Type: 0 = IPMI Messaging session active 1 = Callback Messaging session active 2 = Dial-out Alert active 3 = TAP Page active
	9	Destination Selector for active call-out session. 0 otherwise. [7:4] - reserved [3:0] - Destination selector. Destination 0 is always present as a volatile destination that is used with the Alert Immediate command.
	10:13	If PPP connection: IP address of remote console. (MS-byte first) 00h, 00h, 00h, 00h otherwise.
		The following additional bytes 14:15 are returned if Channel Type = asynch. serial/modem and connection is PPP:
	14:15	Port Address of remote console (LS-byte first). Address that was received in the *Activate Session* command that activated the session.

18.19 Get AuthCode Command

This command is used to send a block of data to the BMC, whereupon the BMC will return a hash of the data together concatenated with the internally stored password for the given channel and user. This command allows a remote console to send an AuthCode and data block to system software on a remote platform, whereby the system software can validate the AuthCode by comparing it with the AuthCode returned by the BMC. This enables the BMC to serve as a validation agent for remote requests that come through local system software instead of through a remote session directly with the BMC.

The application of this command is beyond this specification. However, the following is an outline of potential use of this capability. Remote console software could request that system software perform a particular operation. In response, local system software could deliver a challenge string to the remote console, which would be required to hash it with the desired password and return the AuthCode to the local system software. The local system software would then perform the requested operation only if it found that the AuthCode matched the one returned by the BMC. The local software would typically implement mechanisms to bind the challenge string to the requested operation to ensure that the challenge string and AuthCode combination only applied to a given instance of the requested operation, and even from a particular remote console.

- Managed system delivers a random number token, S, to the Console. In this example, the Console uses S to identify a particular request. The managed system tracks outstanding S values, and expires them either because a valid message was received from a Console that used that token, or because the token was not used within a specified interval.

- Console determines: X = data to be authenticated

  K1 = 16-byte 'signature' of X and a sequence number = hash(X, S, SW_Authentication_Type). Where SW_Authentication_Type is any signature algorithm management software wishes to use for providing a signature given X and S.

  K2 = 16-byte hash of K1 and the password = hash(K1, PWD, Authentication_Type). Where Authentication_Type in this case is one of the supported Authentication Types for the given BMC. *Figure 18-1, AuthCode Algorithms*, specifies how the "Get AuthCode Data" (K1) and password data (PWD) are concatenated for processing according to Authentication_Type. Note that the hash algorithm for K1 does not need to be a BMC supported algorithm or match the algorithm used for K2.

- Console sends X, S, and K2 to software agent on managed system.

- Software agent on the managed system calculates K1 from X and S that it received by locally calculating K1=hash1(X, S, SW_Authentication_Type). The software also verifies that S is a valid outstanding token.

- Managed system passes K1 to BMC. BMC internally looks up password based on the user ID passed in the *Get Authcode* Command and produces: $K2_{BMC}$ = hash(K1, PWD, Authentication_Type)

- Managed system accepts data if software agents finds that K2 = $K2_{BMC}$.

Table 18-21, Get AuthCode Command

	byte	data field
IPMI Request Data	1	Authentication Type [7:4] - reserved [3:0] - hash type 0h = reserved 1h = MD2 2h = MD5 3h = reserved 4h = straight password / key 5h = OEM proprietary all other = reserved
	2	Channel Number [7:4] - reserved [3:0] - Channel number
	3	User ID. (software will typically have to use the Get User Info command to look up the User ID from a username) [7:6] - reserved [5:0] - User ID
	4:19	hash data (must be 16 bytes), or zero padded clear text password if the Authentication Type = straight password
IPMI Response Data	1	Completion Code If authentication type = straight password, BMC returns 'OK' if password was correct for specified user, or error completion code if it is not.
	2:17	AuthCode = See *18.15.1, AuthCode Algorithms*.

18.20 Set Channel Access Command

This command is used to configure whether channels are enabled or disabled, whether alerting is enabled or disabled for a channel, and to set which system modes channels are available under. This configuration is saved in non-volatile storage associated with the BMC. The choice of factory default setting for the non-volatile parameters is left to the implementer or system integrator.

The active (volatile) settings can be overwritten to allow run-time software to make temporary changes to the access. The volatile settings are overwritten from the non-volatile settings whenever the system is reset or transitions to a powered off state.

An implementation can elect to provide a subset of the possible Access Mode options. If a given Access Mode is not supported, the command-specific completion code 83h, access mode not supported, must be returned.

Table 18-22, Set Channel Access Command

	byte	data field
Request Data	1	[7:4] - reserved [3:0] - Channel number
	2	[7:6] - 00b = don't set or change Channel Access 01b = set non-volatile Channel Access according to bits [5:0] 10b = set volatile (active) setting of Channel Access according to bits [5:0] 11b = reserved [5] - PEF Alerting Enable/Disable 0b = enable PEF Alerting 1b = disable PEF Alerting on this channel (the *Alert Immediate* command can still be used to generate alerts) [4] - 0b = enable Per-message Authentication 1b = disable Per-message Authentication. [Authentication required to activate any session on this channel, but authentication not used on subsequent packets for the session.] [3] - User Level Authentication Enable/Disable. Optional. Return a CCh 'invalid data field' error completion code if an attempt is made to set this bit, but the option is not supported. 0b = enable User Level Authentication. All User Level commands are to be authenticated per the Authentication Type that was negotiated when the session was activated. 1b = disable User Level Authentication. Allow User Level commands to be executed without being authenticated. If the option to disable User Level Command authentication is accepted, the BMC will accept packets with Authentication Type set to None if they contain user level commands. For outgoing packets, the BMC returns responses with the same Authentication Type that was used for the request. [2:0] - Access Mode for IPMI messaging (PEF Alerting is enabled/disabled separately from IPMI messaging, see bit 5) 000b = disabled channel disabled for IPMI messaging 001b = pre-boot only channel only available when system is in a powered down state or in BIOS prior to start of boot. 010b = always available channel always available for communication regardless of system mode. BIOS typically dedicates the serial connection to the BMC. 011b = shared same as always available, but BIOS typically leaves the serial port available for software use.

	3	Channel Privilege Level Limit. This value sets the maximum privilege level that can be accepted on the specified channel. [7:6] - 00b = don't set or change channel Privilege Level Limit 01b = set non-volatile Privilege Level Limit according to bits [3:0] 10b = set volatile setting of Privilege Level Limit according to bits [3:0] 11b = reserved [5:4] - reserved [3:0] - Channel Privilege Level Limit 0h = reserved 1h = CALLBACK level 2h = USER level 3h = OPERATOR level 4h = ADMINISTRATOR level 5h = OEM Proprietary level
Response Data	1	Completion Code generic, plus following command-specific completion codes: 82h = set not supported on selected channel (e.g. channel is session-less.) 83h = access mode not supported

18.21 Get Channel Access Command

This command is used to return whether a given channel is enabled or disabled, whether alerting is enabled or disabled for the entire channel, and under what system modes the channel can be accessed.

Table 18-23, Get Channel Access Command

	byte	data field
Request Data	1	[7:4] - reserved [3:0] - Channel number.
	2	[7:6] - 00b = reserved 01b = get non-volatile Channel Access 10b = get present volatile (active) setting of Channel Access 11b = reserved [5:0] - reserved
Response Data	1	Completion Code generic, plus following command-specific completion codes: 82h = Command not supported for selected channel (e.g. channel is session-less.)
	2	[7:6] - reserved [5] - 0b = Alerting enabled 1b = Alerting disabled [4] - 0b = per message authentication enabled 1b = per message authentication disabled [3] - User Level Authentication Enable 0b = User Level Authentication enabled. 1b = User Level Authentication disabled. [2:0] - Access Mode 0h = disabled channel disabled for communication 1h = pre-boot only channel only available when system is in a powered down state or in BIOS prior to start of boot. 2h = always available channel always available for communication regardless of system mode. BIOS typically dedicates the serial connection to the BMC. 3h = shared same as always available, but BIOS typically leaves the serial port available for software use.

3	Channel Privilege Level Limit. This value returns the maximum privilege level that can be accepted on the specified channel. [7:4] - reserved [3:0] - Channel Privilege Level Limit 0h = reserved 1h = CALLBACK level 2h = USER level 3h = OPERATOR level 4h = ADMINISTRATOR level 5h = OEM Proprietary level

18.22 Get Channel Info Command

This command returns media and protocol information about the given channel. The channel protocol may vary with changes to the configuration parameters associated with the channel.

Table 18-24, Get Channel Info Command

IPMI Request Data	1	[7:4] - reserved [3:0] - channel number. Use Eh to get information about the channel this command is being executed from.
IPMI Response Data	1	Completion Code
	2	[7:4] - reserved [3:0] - actual channel number. This value will typically match the channel number passed in the request, unless the request is for channel E, in which case the response returns the actual channel number.
	3	[7] - reserved [6:0] - 7-bit Channel Medium type: per *Table 6-3, Channel Medium Type Numbers*
	4	Channel Protocol Type: [7:5] - reserved [4:0] - 5-bit Channel IPMI Messaging Protocol Type per *Table 6-2, Channel Protocol Type Numbers*
	5	Session support [7:6] - 00b = channel is session-less 01b = channel is single-session 10b = channel is multi-session 11b = channel is session-based (return this value if a channel could alternate between single- and multi-session operation, as can occur with a serial/modem channel that supports connection mode auto-detect) Number of sessions that have been activated on given channel. [5:0] - active session count. 1-based. 00_0000b = no sessions have been activated on this channel.
	6:8	Vendor ID (IANA Enterprise Number) for OEM/Organization that specified the Channel Protocol. Least significant byte first. Returns the IPMI IANA for IPMI-specification defined, non-OEM protocol type numbers other than OEM. The IPMI Enterprise Number is: 7154 (decimal). This gives the values F2h, 1Bh, 00h for bytes 6 through 8, respectively. This value is returned for all channel protocols specified in this document, including PPP.

9:10	Auxiliary Channel Info For Channel = Fh (System Interface) : byte 1: SMS Interrupt Type 00h-0Fh = IRQ 0 through 15, respectively 10h-13h = PCI A-D, respectively 14h = SMI 15h = SCI 20h-5Fh = system interrupt 0 through 63, respectively 60h = assigned by ACPI / Plug 'n Play BIOS FFh = no interrupt / unspecified all other = reserved byte 2: Event Message Buffer Interrupt Type see values for byte 1 For OEM channel types: byte 1:2 = OEM specified per OEM identified by Vendor ID field. All other channel types: byte 1:2 = reserved.

18.23 Set User Access Command

This command is used to configure the privilege level and channel accessibility associated with a given user ID. If this command is not supported, then a single 'null user' (User 1) per channel is assumed and the privilege level and channel access are determined solely by the settings returned by the *Get Channel Access Limits* command. If implemented, this command must support at least the null user (User 1). The number of additional users supported is left to the implementer.

Note: The limits set using the *Set Channel Access* command take precedence over the *Set User Access* command settings. That is, if a given channel is limited to User level then all users will be limited to User level operation regardless of what their User Access levels were set to using the *Set User Access* command.

Note that changes made to the user access and privilege levels may not take affect until the next time the user establishes a session.

Table 18-25, Set User Access Command

	byte	data field
Request Data	1	[7] - 0b = do not change any of the following bits in this byte 1b = enable changing the following bits in this byte [6] - User Restricted to Callback 0b = User Privilege Limit is determined by the User Privilege Limit parameter, below, for both callback and non-callback connections. 1b = User Privilege Limit is determined by the User Privilege Limit parameter for callback connections, but is restricted to Callback level for non-callback connections. Thus, a user can only initiate a Callback when they 'call in' to the BMC, but once the callback connection has been made, the user could potentially establish a session as an Operator. [5] - User Link authentication enable/disable (used to enable whether this user's name and password information will be used for link authentication, e.g. PPP CHAP) for the given channel. Link authentication itself is a global setting for the channel and is enabled/disabled via the serial/modem configuration parameters. 0b = disable user for link authentication 1b = enable user for link authentication [4] - User IPMI Messaging enable/disable (used to enable/disable whether this user's name and password information will be used for IPMI Messaging) 0b = disable user for IPMI Messaging 1b = enable user for IPMI Messaging [3:0] - Channel Number
	2	User ID [7:6] - reserved. [5:0] - User ID. 000000b = reserved.
	3	User Limits [7:4] - reserved [3:0] - User Privilege Limit. (Determines the maximum privilege level that the user is allowed to switch to on the specified channel.) 0h = reserved 1h = Callback 2h = User 3h = Operator 4h = Administrator 5h = OEM Proprietary Fh = NO ACCESS
	(4)	User Session Limit. (Optional) Sets how many simultaneous sessions can be activated with the username associated with this user. If not supported, the username can be used to activate as many simultaneous sessions as the implementation supports. Return a CCh 'invalid data field' error completion code if an attempt is made to set a non-zero value in this field, but the option is not supported. [7:4] - reserved [3:0] - User simultaneous session limit. 1-based. 0h = only limited by the implementations overall support for simultaneous sessions.
Response Data	1	Completion Code. Note: an implementation will not return an error completion code if the user access level is set higher than the privilege limit for a given channel. If it is desired to bring attention to this condition, it is up to software to check the channel privilege limits set using the *Set Channel Access* command and provide notification of any mismatch.

18.24 Get User Access Command

This command is used to retrieve channel access information and enabled/disabled state for the given User ID. The command also returns information about the number of supported users.

Table 18-26, Get User Access Command

	byte	data field
Request Data	1	[7:4] - reserved [3:0] - Channel Number
	2	[7:6] - reserved [5:0] - User ID. 000000b = reserved.
Response Data	1	Completion Code. Note: an implementation will not return an error completion code if the user access level is set higher than the privilege limit for a given channel. If it is desired to bring attention to this condition, it is up to software to check the channel privilege limits and provide notification of the mis-match.
	2	Maximum number of User IDs. 1-based. Count includes User 1. A value of 1 indicates only User 1 is supported. [7:6] - reserved [5:0] - maximum number of user IDs on this channel
	3	Count of currently enabled User IDs (1-based). A value of 0 indicates that all users, including User 1, are disabled. This is equivalent to disabling access to the channel. [7:6] - reserved [5:0] - count of currently enabled user IDs on this channel (Indicates how many User ID slots are presently in use.)
	4	Count of User IDs with fixed names, including User 1(1-based). Fixed names in addition to User 1 are required to be associated with sequential user IDs starting from User ID 2. [7:6] - reserved. [5:0] - count of user IDs with fixed names on this channel
	5	Channel Access [7] - reserved. [6] - 0b = user access available during call-in or callback direct connection 1b = user access available only during callback connection bits 5:4, following, are used for determining the 'count of currently enabled user IDs' in byte 3. Either bit being set to 1b represents an 'enabled user ID'. [5] - 0b = user disabled for link authentication 1b = user enabled for link authentication [4] - 0b = user disabled for IPMI Messaging 1b = user enabled for IPMI Messaging [3:0] - User Privilege Limit for given Channel 0h = reserved 1h = Callback 2h = User 3h = Operator 4h = Administrator 5h = OEM Proprietary Fh = NO ACCESS (Note: this value does not add to, or subtract from, the number of enabled user IDs)

18.25 Set User Name Command

This command allows user names to be assigned to a given User ID. The names are stored as a logical array within non-volatile storage associated with the management controller. Names are stored and retrieved using the User ID as the index into the logical array. There is no configurable name for User ID 1. User ID 1 is reserved for the null user name, User 1.

The management controller does not prevent duplicate usernames from being enabled for the same channel. It is the responsibility of configuration software to ensure that duplicate user names are not enabled simultaneously for the same channel.

Having duplicate usernames will not cause functional problems with the BMC because the BMC will just use the first username match that it finds. However, it could be confusing to the user if they have duplicate usernames enabled for a given channel, since only the settings for the first encountered username would be used by the BMC. See *6.9, Users & Password Support* for more information.

Table 18-27, Set User Name Command

	byte	data field
Request Data	1	User ID [7:6] - reserved. [5:0] - User ID. 000000b = reserved. (User ID 1 is permanently associated with User 1, the null user name).
	2:17	User Name String in ASCII, 16 bytes, max. Strings with fewer than 16 characters are terminated with a null (00h) character and 00h padded to 16 bytes. When the string is read back using the *Get User Name* command, those bytes shall be returned as 0's.
Response Data	1	Completion Code.

18.26 Get User Name Command

This command is used to retrieve user name information that was set using the *Set User Name* command. Configuration software can use this command to retrieve user names.

Table 18-28, Get User Name Command

	byte	data field
Request Data	1	User ID [7:6] - reserved. [5:0] - User ID to return name for. 000000b = reserved.
Response Data	1	Completion Code.
	2:17	User Name String in ASCII, 16 bytes, max. Strings of fewer than 16 characters are returned with null (00h) characters filling in the remaining bytes. BMC does not check to see whether string data is 'printable' or not. Only character that BMC interprets is null (00h).

18.27 Set User Password Command

This command is used to set and change user passwords and to enable and disable User IDs. If no password protection is desired for a given user, the password must be stored as an ASCII null-string. The management controller firmware will force the remaining fifteen bytes to 00h and store the password as sixteen bytes of 00h.

If this command is not supported, then the implication is that only User 1 with a fixed, null password is supported.

The password is stored as a 16-byte 'octet string'. All values (0-255) are allowed for every byte. The management controller does not check the format or interpret values that are passed in with the Set User Password command.

Software is allowed to place additional restrictions on what passwords can be entered, in which case it is the responsibility of configuration software and console software to stay in synch with that definition. For example, remote console software could restrict passwords to the printable ASCII character set in order to simplify direct keyboard entry. If this were done, a companion configuration utility should ensure that the user does not configure the managed system with non-printable passwords. Otherwise, it would be possible for the management controller to be configured with passwords that could not be entered via the remote console utility.

Table 18-29, Set User Password Command

	byte	data field
Request Data	1	User ID. [7:6] - reserved. [5:0] - User ID. 000000b = reserved. (User ID 1 is permanently associated with User 1, the null user name).
	2	[7:2] - reserved [1:0] - operation 00b = disable user 01b = enable user 10b = set password 11b = test password
	3:18	Password data. This is a required, fixed length field when used for the set- and test password operations. If the password is entered as an ASCII string, it must be null (00h) terminated and 00h padded if the string is shorter than 16 bytes. This field need not be present if the operation is 'disable user' or 'enable user'. If this field is present for those operations, the BMC will ignore the data.
Response Data	1	Completion Code. Generic, plus following command-specific completion codes: 80h = password test failed (mandatory)

19. IPMI LAN Commands

This section defines the configuration and control commands that are specific to LAN channels. None of the commands in the following table are required unless a LAN channel is implemented. Refer to *Appendix G - Command Assignments*

for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 19-1, IPMI LAN Commands

Command	Section Defined	O/M
Set LAN Configuration Parameters	19.1	M[1]
Get LAN Configuration Parameters	19.2	M[1]
Suspend BMC ARPs	19.3	O[2]
Get IP/UDP/RMCP Statistics	19.4	O

1. Mandatory if LAN channel is supported.
2. Mandatory if BMC autonomously generates Gratuitous ARPs

19.1 Set LAN Configuration Parameters Command

This command is used for setting parameters such as the network addressing information required for IPMI LAN operation.

Table 19-2, Set LAN Configuration Parameters Command

	byte	data field
Request Data	1	[7:4] - reserved [3:0] - Channel number.
	2	Parameter selector
	3:N	Configuration parameter data, per *Table 19-4, LAN Configuration Parameters*
Response Data	1	Completion Code 80h = parameter not supported. 81h = attempt to set the 'set in progress' value (in parameter #0) when not in the 'set complete' state. (This completion code provides a way to recognize that another party has already 'claimed' the parameters) 82h = attempt to write read-only parameter

19.2 Get LAN Configuration Parameters Command

This command is used for retrieving the configuration parameters from the *Set LAN Configuration* command.

Table 19-3, Get LAN Configuration Parameters Command

	byte	data field
Request Data	1	[7] - 0b = get parameter 1b = get parameter revision only. [6:4] - reserved [3:0] - Channel number.
	2	Parameter selector
	3	Set Selector. Selects a given set of parameters under a given Parameter selector value. 00h if parameter doesn't use a Set Selector.
	4	Block Selector (00h if parameter does not require a block number)
Response Data	1	Completion Code. Generic codes, plus following command-specific completion code(s): 80h = parameter not supported.
	2	[7:0] - Parameter revision. Format: MSN = present revision. LSN = oldest revision parameter is backward compatible with. **11h** for parameters in this specification.
		The following data bytes are not returned when the 'get parameter revision only' bit is 1b.
	3:N	Configuration parameter data, per *Table 19-4, LAN Configuration Parameters* If the rollback feature is implemented, the BMC makes a copy of the existing parameters when the 'set in progress' state becomes asserted (See the Set In Progress parameter #0). While the 'set in progress' state is active, the BMC will return data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the BMC returns parameter data from non-volatile storage.

Table 19-4, LAN Configuration Parameters

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Set In Progress (volatile)	0	data 1 - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software than some other software or utility is in the process of making changes to the data. An implementation can also elect to provide a 'rollback' feature that uses this information to decide whether to 'roll back' to the previous configuration information, or to accept the configuration change. If used, the roll back shall restore all parameters to their previous state. Otherwise, the change shall take effect when the write occurs. [7:2] - reserved [1:0] - 00b = set complete. If a system reset or transition to powered down state occurs while 'set in progress' is active, the BMC will go to the 'set complete' state. If rollback is implemented, going directly to 'set complete' without first doing a 'commit write' will cause any pending write data to be discarded. 01b = set in progress. This flag indicates that some utility or other software is presently doing writes to parameter data. It is a notification flag only, it is not a resource lock. The BMC does not provide any interlock mechanism that would prevent other software from writing parameter data while. 10b = commit write (optional). This is only used if a rollback is implemented. The BMC will save the data that has been written since the last time the 'set in progress' and then go to the 'set in progress' state. An error completion code will be returned if this option is not supported. 11b = reserved
Authentication Type Support (Read Only)	1	This 'read only' field returns which possible Authentication Types (algorithms) can be enabled for the given channel. The following Authentication Type Enables parameter selects which Authentication Types are available when activating a session for a particular maximum privilege level. [7:6] - reserved [5:0] - Authentication type(s) enabled for this channel (bitfield): All bits: 1b = supported 0b = authentication type not available for use. [5] - OEM proprietary (per OEM identified by the IANA OEM ID in the RMCP Ping Response) [4] - straight password / key [3] - reserved [2] - MD5 [1] - MD2 [0] - none

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Authentication Type Enables	2	This field is used to configure which Authentication Types are available for use when a remote console activates an IPMI messaging connection to the BMC for a given requested maximum privilege level. Once the session has been activated, the accepted authentication type will be the only one used for *authenticated* packets, regardless of the present operating privilege level, or the privilege level associated with the command. Depending on configuration of per-message and user-level authentication disables, unauthenticated packets (authentication type = none) may also be accepted. The BMC makes no attempt to check or ensure that stricter authentication types are associated with higher requested maximum privilege levels. E.g. it is possible to configure the BMC so activating a session with a maximum privilege level of 'User' requires MD5 while 'Admin' requires 'none'. Note: An implementation that has fixed privilege and authentication type assignments, in which case this parameter can be implemented as Read Only. It is recommended that an implementation that implements a subset of the possible authentication types returns a CCh error completion code if an attempt is made to select an unsupported authentication type. byte 1: Authentication Types returned for maximum requested privilege = **Callback** level. [7:6] - reserved [5:0] - Authentication type(s) enabled for this channel (bitfield): All bits: 1b = authentication type enabled for use at given privilege level 0b = authentication type not available for use at given privilege level. [5] - OEM proprietary (per OEM identified by the IANA OEM ID in the RMCP Ping Response) [4] - straight password / key [3] - reserved [2] - MD5 [1] - MD2 [0] - none byte 2: Authentication Type(s) for maximum privilege = **User** level (format follows byte 1) byte 3: Authentication Type (s) for maximum privilege = **Operator** level (format follows byte 1) byte 4: Authentication Type (s) for maximum privilege = **Administrator** level (format follows byte 1) byte 5: Authentication Type (s) for maximum privilege = **OEM** level (format follows byte 1)
IP Address	3	data 1:4 - IP Address MS-byte first.
IP Address Source	4	data 1 [7:4] - reserved [3:0] - address source 0h = unspecified 1h = static address (manually configured) 2h = address obtained by BMC running DHCP 3h = address loaded by BIOS or system software 4h = address obtained by BMC running other address assignment protocol
MAC Address	5	data 1:6 - MAC Address for messages transmitted from BMC. MS-byte first.
Subnet Mask	6	data 1:4 - Subnet Mask. MS-byte first.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
IPv4 Header Parameters	7	data 1 - Time-to-live. 1-based. (Default = 40h) Value for time-to-live parameter in IP Header for RMCP packets and PET Traps transmitted from this channel. data 2 [7:5] - Flags. Sets value of bit 1 in the Flags field in the IP Header for packets transmitted by this channel. (Default = 010b "don't fragment") [4:0] - reserved data 3 [7:5] - Precedence (Default = 000b) [4:1] - Type of Service (Default = 1000b, "minimize delay") [0] - reserved
Primary RMCP Port Number (optional)	8	data 1:2 - Primary RMCP Port Number, LSByte first. Default = 26Fh (RMCP 'Aux Bus Shunt' port)
Secondary RMCP Port Number (optional)	9	data 1:2 - Secondary Port Number, LSByte first. Default = 298h (RMCP 'Secure Aux Bus' port)
BMC-generated ARP control (optional[2])	10	data 1 - BMC-generated ARP control. Note: the individual capabilities for BMC-generated ARP responses and BMC-generated Gratuitous ARPs are individually optional. The BMC should return an error completion code if an attempt is made to enable an unsupported capability. [7:2] - reserved [1] - 1b = enable BMC-generated ARP responses 0b = disable BMC-generated ARP responses [0] - 1b = enable BMC-generated Gratuitous ARPs 0b = disable BMC-generated Gratuitous ARPs
Gratuitous ARP interval (optional)	11	data 1 - Gratuitous ARP interval Gratuitous ARP interval in 500 millisecond increments. 0-based. Interval accuracy is +/- 10%. If this configuration parameter is not implemented, gratuitous ARPs shall be issued at a rate of once every 2 seconds.
Default Gateway Address	12	data 1:4 - IP Address MS-byte first. This is the address of the gateway (router) used when the BMC sends a message or alert to a party on a different subnet than the one the BMC is on.
Default Gateway MAC Address	13	data 1:6 - MAC Address. MS-byte first.
Backup Gateway Address	14	data 1:4 - IP Address MS-byte first. This is the address of an alternate gateway (router) that can be selected when a sending a LAN Alert.
Backup Gateway MAC Address	15	data 1:6 - MAC Address. MS-byte first.
Community String	16	data 1:18 - Community String Default = 'public'. Used to fill in the 'Community String' field in a PET Trap. This string may optionally be used to hold a vendor-specific string that is used to provide the network name identity of the system that generated the event. Printable ASCII string-. If a full 18 non-null characters are provided, the last character does not need to be a null. 18 characters must be written when setting this parameter, and 18 will be returned when this parameter is read. The null character, and any following characters, will be ignored when the Community String parameter is placed into the PET. The BMC will return whatever characters were written. I.e. it will not set bytes following the null to any particular value.
Number of Destinations (Read Only)	17	data 1 - Number of LAN Alert Destinations supported on this channel. (Read Only). At least one set of non-volatile destination information is required if LAN alerting is supported. Additional non-volatile destination parameters can optionally be provided for supporting an alert 'call down' list policy. A maximum of ***fifteen*** (1h to Fh) non-volatile destinations are supported in this specification. Destination 0 is always present as a volatile destination that is used with the *Alert Immediate* command. [7:4] - reserved. [3:0] - Number LAN Destinations. A count of 0h indicates LAN Alerting is not supported.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Destination Type (volatile / non-volatile - see description)	18	Sets the type of LAN Alert associated with the given destination. This parameter is not present if the Number of Destinations parameter is 0. data 1 - Set Selector = Destination selector, 0 based. [7:4] - reserved [3:0] - Destination selector. Destination 0 is always present as a volatile destination that is used with the *Alert Immediate* command. data 2 - Destination Type [7] - Alert Acknowledge. 0b = Unacknowledged. Alert is assumed successful if transmission occurs without error. This value is also used with Callback numbers. 1b = Acknowledged. Alert is assumed successful only if acknowledged is returned. Note, some alert types, such as Dial Page, do not support an acknowledge. [6:3] - reserved [2:0] - Destination Type 000b = PET Trap destination 001b - 101b = reserved 110b = OEM 1 111b = OEM 2 data 3 - Alert Acknowledge Timeout / Retry Interval, in seconds, 0-based (i.e. minimum timeout = 1 second). This value sets the timeout waiting for an acknowledge, or the time between automatic retries depending on whether the alert is acknowledge or not. Recommended factory default = 3 seconds. Value is ignored if alert type does not support acknowledge, or if the Alert Acknowledge bit (above) is 0b. data 4 - Retries [7:4] - reserved [3] - reserved [2:0] - Number of times to retry alert to given destination. 0 = no retries (alert is only sent once). If the alert is acknowledged (Alert Acknowlege bit = 1b) the alert will only be retried if a timeout occurs waiting for the acknowledge. Otherwise, this value selects the number of times an unacknowledged alert will be sent out. The timeout interval or time between retries is set by the Alert Acknowledge Timeout / Retry Interval value (byte 3 of this parameter).
Destination Addresses	19	Sets/Gets the list of IP addresses that a LAN alert can be sent to. This parameter is not present if the Number of Destinations parameter is 0. data 1 - Set Selector = Destination Selector. [7:4] - reserved [3:0] - Destination selector. Destination 0 is always present as a volatile destination that is used with the *Alert Immediate* command. data 2 - Address Format [7:4] - Address Format. 0h = IPv4 IP Address followed by DIX Ethernet/802.3 MAC Address [3:0] - reserved For Address Format = 0h: data 3 - Gateway selector [7:1] - reserved [0] - 0b = use default gateway 1b = use backup gateway data 4:7 - Alerting IP Address (MS-byte first) data 8:13 - Alerting MAC Address (MS-byte first)
OEM Parameters	192 : 255	This range is available for special OEM configuration parameters. The OEM is identified according to the Manufacturer ID field returned by the *Get Device ID* command.

1. Choice of system manufacturing defaults is left to the system manufacturer unless otherwise specified.
2. This configuration parameter must be supported if the controller autonomously issues gratuitous ARPs or ARP responses.

19.3 Suspend BMC ARPs Command

This command can be used to suspend BMC-generated Gratuitous ARPs or ARP responses (if implemented and enabled) while run-time software is handling them. ARPs will automatically resume and the ARP suspend option will be automatically disabled whenever the watchdog timer stops or when the *Set Watchdog Timer* command is executed. Software must explicitly enable the suspension after starting or re-starting the Watchdog Timer. This is to ensure that BMC-generated ARPs will resume if the watchdog expires (indicating that software may no longer be handling the ARPs).

Software can examine the LAN Configuration Parameters for the desired channel to see if Gratuitous ARPs or BMC-generated ARP Responses are enabled.

Table 19-5, Suspend BMC ARPs Command

	byte	data field
Request Data	1	[7:4] reserved [3:0] Channel number
	2	[7:2] - reserved ARP Response suspend [1] - 0b = do not suspend BMC-generated ARP responses while the Watchdog Timer is running. 1b = suspend BMC-generated ARP responses while Watchdog Timer is running. This value must be set after the Watchdog has been configured using the Set Watchdog Timer command and, if it is desired that the suspension of ARPs be continued, must be re-written after any subsequent Set Watchdog Timer commands. Gratuitous ARP suspend [0] - 0b = do not suspend BMC-generated Gratuitous ARPs while the Watchdog Timer is running. 1b = suspend BMC-generated Gratuitous ARPs while the Watchdog Timer is running. This value must be set after the Watchdog has been configured using the Set Watchdog Timer command and, if it is desired that the suspension of ARPs be continued, must be re-written after any subsequent Set Watchdog Timer commands.
Response Data	1	Completion Code
	2	Present state of ARP suspension: [7:2] - reserved ARP Response status [1] - 1b = BMC-generated ARP Responses are occurring 0b = BMC-generated ARP Responses are presently being suspended or are disabled. Gratuitous ARP Response status [0] - 1b = BMC-generated Gratuitous ARPs are occurring 0b = BMC-generated Gratuitous ARPs are presently being suspended or are disabled.

19.4 Get IP/UDP/RMCP Statistics Command

This command is used retrieving information about the IP connections on the given channel. The info is cumulative but volatile. I.e. it is not required to keep these statistics across management controller power cycles. It is recommended that this information be kept across system resets and power cycles. The statistics values are initialized to 0 unless otherwise noted.

Table 19-6, Get IP/UDP/RMCP Statistics Command

	byte	data field
Request Data	1	[7:4] reserved [3:0] Channel number.
	2	Clear Statistics [7:1]- reserved [0] - 0b = don't clear statistics 1b = clear all statistics values to 0
Response Data	1	Completion Code
	2:3	IP Packets Received. All statistics returned by this command are 1-based unless otherwise noted. All statistics stop accumulating at FFFFh unless otherwise noted.
	4:5	Received IP Header Errors
	6:7	Received IP Address Errors
	8:9	Fragmented IP Packets Received
	10:11	IP Packets Transmitted
	12:13	UDP Packets Received
	14:15	Valid RMCP Packets Received
	16:17	UDP Proxy Packets Received
	18:19	UDP Proxy Packets dropped

20. IPMI Serial/Modem Commands

This section defines the configuration and control commands that are specific to serial/modem channels. None of the commands in the following table are required unless a serial/modem channel is implemented. Refer to *Appendix G - Command Assignments*

for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 20-1, IPMI Serial/Modem Commands

Command	Section Defined	O/M
Set Serial/Modem Configuration	20.1	M[1]
Get Serial/Modem Configuration	20.2	M[1]
Set Serial/Modem Mux	20.3	O[2]
Get TAP Response Codes	20.4	O[3]
Set PPP UDP Proxy Transmit Data	20.5	O[4]
Get PPP UDP Proxy Transmit Data	20.6	O[4]
Send PPP UDP Proxy Packet	20.7	O[4]
Get PPP UDP Proxy Receive Data	20.8	O[4]
Serial/Modem Connection Active	20.9	M[1]
Callback	20.10	O
Set User Callback Options	20.11	O[5]
Get User Callback Options	20.12	O[5]

1. Mandatory if serial/modem channel(s) supported.
2. Mandatory if Serial Port Sharing is supported.
3. Mandatory if TAP Paging is supported. If TAP Paging is supported it is recommended, but not mandatory, that it be supported on all serial/modem channels that could support a modem connection. (Some serial/modem channels may never be connected to a modem)
4. Mandatory if PPP UDP Proxy capability is supported.
5. Mandatory if IPMI Callback is supported. Note that CBCP callback support is optional. Whether CBCP is supported or not is determined from the serial/modem configuration parameters.

20.1 Set Serial/Modem Configuration Command

This command is used for setting parameters such as the string used for initializing the modem, communication bit rates, and selecting configuration options such as Direct Connect versus Modem Connect.

Table 20-2, Set Serial/Modem Configuration Command

	byte	data field
Request Data	1	[7:4] - reserved [3:0] - Channel number.
	2	Parameter selector
	3:N	Configuration parameter data, per *Table 20-4, Serial/Modem Configuration Parameters*
Response Data	1	Completion Code. Generic plus the following command-specific completion codes: 80h = parameter not supported. 81h = attempt to set the 'set in progress' value (in parameter #0) when not in the 'set complete' state. (This completion code provides a way to recognize that another party has already 'claimed' the parameters) 82h = attempt to write read-only parameter 83h = attempt to read write-only parameter

20.2 Get Serial/Modem Configuration Command

This command is used for retrieving the configuration parameters from the *Set Serial/Modem Configuration* command.

Table 20-3, Get Serial/Modem Configuration Command

	byte	data field
Request Data	1	[7] - 0b = get parameter 1b = get parameter revision only [6:4] - reserved [3:0] - Channel number.
	2	Parameter selector
	3	Set Selector. Selects a particular set or block data under the given parameter selector. 00h if parameter does not use a set selector.
	4	Block Selector (00h if parameter does not require a block number)
Response Data	1	Completion Code. Generic plus the following command-specific completion codes: 80h = parameter not supported.
	2	[7:0] - Parameter revision. Format: MSN = present revision. LSN = oldest revision parameter is backward compatible with. **11h** for parameters in this specification.
		The following data bytes are not returned when the 'get parameter revision only' bit is 1b.
	3:N	Configuration parameter data, per *Table 20-4, Serial/Modem Configuration Parameters* If the rollback feature is implemented, the BMC makes a copy of the existing parameters when the 'set in progress' state becomes asserted (See the Set In Progress parameter #0). While the 'set in progress' state is active, the BMC will return data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the BMC returns parameter data from non-volatile storage.

Table 20-4, Serial/Modem Configuration Parameters

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Set In Progress (volatile)	0	data 1 - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software than some other software or utility is in the process of making changes to the data. An implementation can also elect to provide a 'rollback' feature that uses this information to decide whether to 'roll back' to the previous configuration information, or to accept the configuration change. If used, the roll back shall restore all parameters to their previous state. Otherwise, the change shall take effect when the write occurs. [7:2] - reserved [1:0] - 00b = set complete. If a system reset or transition to powered down state occurs while 'set in progress' is active, the BMC will go to the 'set complete' state. If rollback is implemented, going directly to 'set complete' without first doing a 'commit write' will cause any pending write data to be discarded. 01b = set in progress. This flag indicates that some utility or other software is presently doing writes to parameter data. It is a notification flag only, it is not a resource lock. The BMC does not provide any interlock mechanism that would prevent other software from writing parameter data while. 10b = commit write (optional). This is only used if a rollback is implemented. The BMC will save the data that has been written since the last time the 'set in progress' and then go to the 'set in progress' state. An error completion code will be returned if this option is not supported. 11b = reserved
Authentication Type Support (Read Only)	1	This 'read only' field returns which possible Authentication Types (algorithms) can be enabled for the given channel. The following Authentication Type Enables parameter selects which Authentication Types are available when activating a session for a particular maximum privilege level. [7:6] - reserved [5:0] - Authentication type(s) enabled for this channel (bitfield): All bits: 1b = supported 0b = authentication type not available for use. [5] - OEM proprietary (per OEM identified by the IANA OEM ID in the RMCP Ping Response) [4] - straight password / key [3] - reserved [2] - MD5 [1] - MD2 [0] - none

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Authentication Type Enables	2	This field is used to configure which Authentication Types are available for use when a remote console activates an IPMI messaging connection to the BMC for a given requested maximum privilege level. Once the session has been activated, the accepted authentication type will be the only one used for *authenticated* packets, regardless of the present operating privilege level, or the privilege level associated with the command. Depending on configuration of per-message and user-level authentication disables, unauthenticated packets (authentication type = none) may also be accepted. The BMC makes no attempt to check or ensure that stricter authentication types are associated with higher requested maximum privilege levels. E.g. it is possible to configure the BMC so activating a session with a maximum privilege level of 'User' requires MD5 while 'Admin' requires 'none'. Note: An implementation that has fixed privilege and authentication type assignments, in which case this parameter can be implemented as Read Only. It is recommended that an implementation that implements a subset of the possible authentication types returns a CCh error completion code if an attempt is made to select an unsupported authentication type. byte 1: Authentication Types returned for maximum requested privilege = **Callback** level. [7:6] - reserved [5:0] - Authentication type(s) enabled for this channel (bitfield): All bits: 1b = authentication type enabled for use at given privilege level 0b = authentication type not available for use at given privilege level. [5] - OEM proprietary (For PPP, per OEM identified by the IANA OEM ID in the RMCP Ping Response. For other serial/modem modes, a-priori knowledge of the algorithm is required.) [4] - straight password / key [3] - reserved [2] - MD5 [1] - MD2 [0] - none byte 2: Authentication Type(s) for maximum privilege = **User** level (format follows byte 1) byte 3: Authentication Type (s) for maximum privilege = **Operator** level (format follows byte 1) byte 4: Authentication Type (s) for maximum privilege = **Administrator** level (format follows byte 1) byte 5: Authentication Type (s) for maximum privilege = **OEM** level (format follows byte 1)

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Connection Mode	3	data 1 - connection mode - This parameter determines the protocols used when performing IPMI messaging to the BMC. [7] - 0b = Modem Connect mode 1b = Direct Connect mode [6] - reserved Connection mode enables. Sets which mode or modes can be used for establishing an IPMI Messaging connection with the BMC. If more than mode is enabled, the BMC will attempt to auto-detect the appropriate connection mode based on snooping traffic from the remote console. Supporting connection mode auto-detect is *optional*. If an implementation does not support the capability, it shall return an "Illegal Data Field" completion code (CCh) if an attempt is made to enable more than one connection mode at a time. An 'Illegal Data Field' code shall also be returned if an attempt is made to enable a connection mode that the implementation does not support. [5:3] - reserved [2] - 1b = enable Terminal mode (Note: Terminal mode auto-detect also requires that the "Enable Baseboard-to-BMC switch on <ESC>(" option be enabled in the Mux Switch Configuration parameters, below.) [1] - 1b = enable PPP mode [0] - 1b = enable Basic mode
Session Inactivity Timeout (optional)	4	[7:4] - reserved [3:0] - Inactivity timeout in 30 second increments. 1-based. 0h = session does not timeout and close due to inactivity.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Channel Callback Control	5	This is parameter determines which callback options are enabled or disabled for the channel. These parameters take precedence over any user-specific callback settings configured using the *Set User Callback Options* command. An option must be enabled in this global parameter in order to be able to be enabled in the user-specific callback settings. (see *20.11, Set User Callback Options Command*). data 1 - callback enable [7:2] - reserved [1] - 1b = enable CBCP callback protocol (see *13.6.1, Callback Control Protocol* (CBCP) Support) [0] - 1b = enable IPMI callback data 2 - CBCP Negotiation Options. [7:4] - reserved. [3] - 1b = enable callback to one from list of possible numbers [2] - 1b = enable user-specifiable callback number. Allow caller to specify number to be used for callback. [1] - 1b = enable Pre-specified number. Allow caller to request that callback occur to a single, pre-specified number for the user. [0] - 1b = enable No Callback. Allow caller to request that callback not be used. data 3 Callback destination 1. This field holds a Destination Selector that picks which Destination Dial String from the serial/modem configuration parameters to use for callback. This selector is used when the 'pre-specified number' option is used. Otherwise, this is the first number in the list when the "caller selects one number from a list of numbers" option is used. Refer to *13.6.1, Callback Control Protocol* (CBCP) Support for characters supported in dial strings for CBCP. FFh = unspecified. Note, if this field is set to FFh, the BMC should reject negotiation for the 'pre-specified number' option, even if it is enabled in the CBCP Negotiation Options field, above. data 4 Callback destination 2. This is the second number in the list when the "caller selects one number from a list of numbers" option is used. FFh = unspecified. Note, at least one destination must be specified in order for the 'callback to one from a list of numbers' option to be negotiated, even it that option is enabled in the CBCP Negotiation Options field, above. data 5 Callback destination 3. This is the third number in the list when the "caller selects one number from a list of numbers" option is used. FFh = unspecified. Note, at least one destination must be specified in order for the 'callback to one from a list of numbers' option to be negotiated, even it that option is enabled in the CBCP Negotiation Options field, above.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Session Termination	6	data 1 - connection termination. This parameter determines whether serial/modem connections are terminated by inactivity or by a loss of DCD. For modem mode, the line is hung-up when the specified termination condition occurs. For both modem and direct connect mode, the session will be terminated and will need to be reactivated and authenticated (if authentication is enabled) in order for IPMI messaging communications to be re-established. [7:2] - reserved [1] - 1b = enable session inactivity timeout 0b = disable session inactivity timeout [0] - 1b = close session on loss of DCD (this should be used as the default setting for both Modem Connect and Direct Connect mode) [Also see bit to enable mux switch on DCD assertion, in Mux Switch Control parameter, below] 0b = ignore DCD (DCD is never ignored in Modem Mode)
IPMI Messaging Comm Settings	7	This parameter is used for IPMI messaging in PPP Mode, Basic Mode, and Terminal Mode. These settings can be overridden on a per-destination basis for Dial-out LAN Alerting, Dial-Paging, TAP Paging, and Callback Security, according to the Destination Comm Settings parameter, below. IPMI Messaging always occurs with 8 bits/character, no parity, and 1 stop bit. data 1 - flow control, DTR hang-up, asynch format [7:6] - Flow control 00b = No flow control 01b = RTS/CTS flow control (a.k.a. hardware handshake) 10b = XON/XOFF flow control (optional) [if implemented, may not be supported for all connection modes] 11b = Reserved. [5] - DTR hang-up 0b = disable DTR hang-up 1b = enable DTR hang-up [4:0] - reserved. data 2 - bit rate [7:4] - reserved [3:0] - 0-5h = reserved. Support for bit rates other than 19.2 kbps is optional. The BMC must return an error completion if a requested bit rate is not supported. It is recommended that the 'parameter out-of-range' (C9h) code be used for this situation. 6h = 9600 bps 7h = 19.2 kbps (required) 8h = 38.4 kbps 9h = 57.6 kbps Ah = 115.2 kbps

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Mux Switch Control	8	data 1 See *13.2.4, Serial Port Switching* for additional information on these bits. Bit [3] is only applicable if PPP Mode is supported. [7] - reserved [6] - 0b = Disable system power-up/wakeup via [MSVT] <ESC>^ escape sequence 1b = Enable system power-up/wakeup via [MSVT] escape sequence[3] [5] [5] - 0b = Disable hard reset on [MSVT] <ESC>R<ESC>r<ESC>R escape sequence 1b = Enable hard reset on [MSVT] escape sequence[3] [4] - 0b = Disable Baseboard-to-BMC switch on detecting basic mode *Get Channel Authentication Capabilities* message pattern in serial stream. 1b = Enable Baseboard-to-BMC switch on detecting basic mode *Get Channel Authentication Capabilities* message pattern in serial stream. [3] - 0b = Disable switch to BMC on PPP IPMI-RMCP pattern 1b = Enable switch on PPP IPMI-RMCP pattern [2] - 0b = Disable BMC-to-Baseboard switch on [MSVT] <ESC>Q 1b = Enable BMC-to-Baseboard switch on [MSVT] <ESC>Q[3] [1] - 0b = Disable Baseboard-to-BMC switch on [MSVT] <ESC>(1b = Enable Baseboard-to-BMC switch on [MSVT]<ESC>([3] [5] [0] - Following only used in Direct Connect Mode (ignored in Modem Mode) 0b = Disable mux switch to BMC on DCD loss 1b = Enable mux switch on DCD loss data 2 [7:4] - reserved [3] - 0b = Disable Serial Port Sharing. (cannot force mux setting via *Set Serial/Modem Mux* command) The serial connection is assigned to the BMC whenever the channel is enabled, and cannot be switched to the baseboard UART. Note: if this setting is 0b and the serial/modem channel is disabled, the mux will be connected to the baseboard UART and will not be able to be switched to the BMC by IPMI command. If Serial Port Sharing is not implemented, this bit will always be set to 'disabled' and will not be changeable. 1b = Enable Serial Port Sharing (can force mux setting using *Set Serial/Modem Mux* command) [2] - 0b = Disable Serial/Modem Connection Active message during Callback connection. 1b = Enable *Serial/Modem Connection Active* message during Callback connection. [1] - 0b = Disable *Serial/Modem Connection Active* message during direct-call 1b = Enable *Serial/Modem Connection Active* message during direct-call [0] - 0b = Send *Serial/Modem Connection Active* message only once before switching mux to system 1b = Mux switch acknowledge. Retry *Serial/Modem Connection Active* message with retry counts and interval as specified in Section *13.3.2, Mux Switch Coordination*.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Modem Ring Time	9	Configures the amount of time that the BMC needs to see transitions or an active state on RI before the BMC claims the mux in Modem Mode. This setting only applies when the Access Mode is set to “Shared” or “Pre-boot Only”, Serial Port Sharing is enabled, the channel is enabled for IPMI Messaging. This includes when the system is powered down, in order to allow the possibility for using “Wake On Ring” to trigger a wake of the system without causing the BMC answering the phone. See *13.2.7, Serial Port Sharing Access Characteristics* for additional information. data 1 - Ring Duration [7:6] - reserved [5:0] - Ring duration in 500 ms increments. 1 based. 00_0000b = BMC switches mux immediately on first detected transition of RI. 11_1111b (3Fh) = reserved data 2 - Ring Dead Time [7:4] - reserved [3:0] - Amount of time, in 500 ms increments, that the RI signal must be deasserted before the BMC determines that ringing has stopped. 0h = 500 ms.
Modem Init String	10	Sets the modem initialization string data. The BMC automatically follows this string with an <enter> character when sending it to the modem. data 1 - set selector = 16-byte block number to set, 1 based. Two blocks required, at least three recommended. data 2:N - Modem Init string data. String is stored as null terminated ASCII string.
Modem Escape Sequence (optional)	11	data1:5- Null terminated ASCII string for the Escape string to be sent to the modem. If this parameter is empty, or this configuration option is not implemented, the default ‘+++’ sequence will be used. [If a full five characters are provided, the last character does not need to be null]
Modem Hang-up Sequence (optional)	12	data1:8 - Null terminated ASCII string for the hang-up string to be sent to the modem. The BMC automatically follows this string with an <enter> character when sending it to the modem. If this parameter is empty, or this configuration option is not implemented, the default ‘ATH’ sequence will be used. [If a full eight characters are provided, the last character does not need to be null]
Modem Dial Command (optional)	13	data1:8 - Null terminated ASCII string for the modem string used to initiate a dial sequence with the modem. If this parameter is empty, or this configuration option is not implemented, the default ‘ATD’ sequence will be used. [If a full eight characters are provided, the last character does not need to be null]
Page Blackout Interval	14	data 1 - Dial Page, Directed Alert, or TAP Blackout Interval in minutes. 1 based. 00h = no blackout. See *Section 13.10, Page Blackout Interval* for more information.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Community String	15	data 1:18 - Community String Default = 'public'. Used to fill in the 'Community String' field in a PET format trap. This string may optionally be used to hold a vendor-specific string that is used to provide the network name identity of the system that generated the event. Printable ASCII string. If 18 non-null characters are provided, the last character does not need to be a null. 18 characters must be written when setting this parameter, and 18 will be returned when this parameter is read. The null character, and any following characters, will be ignored when the Community String parameter is placed into the PET. The BMC will return whatever characters were written. I.e. it will not set bytes following the null to any particular value. (Community strings are supported on a 'per channel' basis in order to allow the possibility that a different Community String would be used based on the type of connection.)
Number of Alert Destinations (READ ONLY)	16	data 1 - Number of non-volatile Alert Destinations for this channel. Destination 0 is always present as a volatile destination that is used with the *Alert Immediate* command. [7:5] - reserved. [3:0] - Number of non-volatile alert destinations. One minimum, ***fifteen*** non-volatile destinations maximum. It is recommended that an implementation provide at least two destination numbers for each page/alert type supported, plus two for callback if callback is supported. 0h = Page Alerting not supported.
Destination Info (volatile) & (non-volatile) - see description.	17	Sets the type of page associated with the given destination. For Dial Page, TAP Page, and Callback, this also selects the dial string associated with the destination. Destination 0 is used to set a temporary, RAM-based, value. This value is used with the *Alert Immediate* command. The value is not guaranteed to be retained across BMC or system hard resets or power on/off transitions. data 1 - Destination Selector A minimum of one and a maximum of ***fifteen*** non-volatile destinations are supported in the specification. If callback is supported, the callback number is also a type of destination. Destination 0 is always present as a volatile destination that is used with the *Alert Immediate* command. [7:4] - reserved [3:0] - destination selector. 0h = volatile destination. 1-Fh = non-volatile destination. data 2 - Destination Type [7] - Alert Acknowledge. Note, some alert types, such as Dial Page, do not support acknowledge, in which case this bit is ignored and should be written as 0b. 0b = Unacknowledged. Alert is assumed successful if transmission occurs without error. This value is also used with Callback numbers. 1b = Acknowledged. Alert is assumed successful only if acknowledged is returned. [6:4] - reserved [3:0] - Destination Type: 0000b = Dial Page 0001b = TAP Page 0010b = PPP Alert (PET Alert delivered via a PPP-to-LAN connection) 0011b = Basic Mode Callback 0100b = PPP Mode Callback 0101b:1101b = reserved

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
		1110b = OEM 1 1111b = OEM 2 data 3 - Alert Acknowledge Timeout, in seconds, 0-based (i.e. minimum timeout = 1 second). Recommended factory default = 5 seconds. Value is ignored if alert type does not support acknowledge, or if the Alert Acknowledge bit (above) is 0b. data 4: Retries [7] - reserved [6:4] - Number of times to retry alert once call connection has been made. (Does not apply to TAP Page or Dial Page alerts) 1-based. 000b = no retries (alert is only sent once). [3] - reserved [2:0] - Number of times to retry *call* to given destination. (See below for Call Retry Interval parameter) 1-based. 000b = no retries (call is only tried once). data 5: Destination Type Specific: For Destination Type = Dial Page: [7:4] - Dial String Selector [3:0] - reserved For Destination Type = TAP Page: Indicates which set of TAP Service Settings should be used for communication with this destination. [7:4] - reserved [3:0] - TAP Account Selector For Destination Type = PPP Alert: Indicates which set of PPP Account settings should be used for communication with the selected destination. [7:4] - Destination IP Address Selector [3:0] - PPP Account Set Selector For Destination Type = PPP Mode Callback or Basic Mode Callback: [7:4] - = Destination IP Address Selector for PPP Mode Callback (The IP Address is used to enable the BMC to send a Serial/Modem Connection Active message once the connection has been established.) = Dial String Selector for Basic Mode Callback [3:0] - PPP Account Set Selector (PPP Mode Callback only, reserved otherwise)
Call Retry Interval (non-volatile)	18	[7:0] - Number of seconds between call ('busy signal') retries.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Destination Comm Settings (volatile) & (non-volatile) - see description.	19	data 1 - Destination Selector Note that each destination has its own comm settings. [7:4] - reserved [3:0] - Destination Selector. 0 = volatile destination. 1-Fh = non-volatile destination. Destination comm settings. These settings override the IPMI Messaging Comm Setting configuration parameter. data 2 - flow control, DTR hang-up, asynch format [7:6] - flow control 00b = No flow control 01b = RTS/CTS flow control 10b = XON/XOFF flow control 11b = reserved [5] - reserved [4] - stop bits 0b = 1 stop bit (default) 1b = 2 stop bits [3] - character size 0b = 8 bits (must be 8-bit for PPP) 1b = 7-bits (most TAP services use 7-bit) [2:0] - parity 000b = no parity. 001b = odd parity. 010b = even parity data 3 - bit rate [7:4] - reserved [3:0] - bit rate 0-5h = reserved 6h = 9600 bps 7h = 19.2 kbps (required) 8h = 38.4 kbps 9h = 57.6 kbps Ah = 115.2 kbps
Number of Dial Strings (READ ONLY)	20	data 1 - Number of non-volatile Dial Strings for this channel. Dial String 0 is always present and is typically used as a volatile destination that is used with the *Alert Immediate* command. [7:5] - reserved. [3:0] - Number of non-volatile dial strings. One minimum, ***fifteen*** non-volatile dial strings maximum. An implementation should support one dial string for each destination. 0h = Serial/Modem Alerting and Callback not supported.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Destination Dial Strings (volatile) & (non-volatile) - see description.	21	Sets the phone number that the page, alert is to be sent to. The BMC automatically precedes this string with the Modem Init String sequence, when not using direct connect mode. The string can contain embedded modem control sequence characters. data 1 - destination selector [7:4] - reserved [3:0] - Dial String Selector. 0 = volatile dial string 1-Fh = non-volatile dial string. data 2 - block number to set, 1 based. Blocks are 16-bytes. At least two blocks are required per number, supporting a dial string of 31 characters plus terminator. data 3:N - Dial string data. Null terminated ASCII string.
Number of Alert Destination IP Addresses (READ ONLY)	22	data 1 - Number of non-volatile Alert Destination IP Addresses for this channel. Address 0 is always present and is typically used as a volatile destination that is used with the *Alert Immediate* command. It is recommended that there be at least one destination IP Address per PPP Account. [7:5] - reserved. [3:0] - Number of Destination IP Addresses. 0h = PPP Alerting and Callback are not supported.
Destination IP Addresses	23	data 1 - destination selector [7:4] - reserved [3:0] - Destination IP Address Selector. 0 = volatile IP Address location 1-Fh = non-volatile IP Address data 2:5 - destination IP Address. MS-byte first.
Number of TAP Accounts (READ ONLY)	24	data 1 - Number of non-volatile TAP Accounts for this channel. Account 0 is always present and is typically used as a volatile destination that is used with the *Alert Immediate* command. It is not included in the count. [7:5] - reserved. [3:0] - Number of TAP Accounts. 0h = TAP not supported.
TAP Account	25	data 1 - set selector = TAP Account Selector, 1-based. At least one set of TAP Account parameters must be provided for each TAP destination supported. Account 0 is always present and is typically used as a volatile destination that is used with the *Alert Immediate* command. data 2 - TAP Dial String and Service Setting selectors [7:4] - Dial String Selector [3:0] - TAP Service Settings Selector. 1-based. 0h if Destination Type is not 'TAP Page'
TAP Passwords (WRITE ONLY)	26	data 1 - set selector = TAP Account selector, 1 based. data 2:8 - Password. This string is up to six ASCII characters. Null terminated if fewer than six characters are used.
TAP Pager ID Strings	27	This parameter sets and returns the TAP Pager ID (also referred to as 'Field 1') for the specified destination. This typically holds the phone number of the party to be paged. Note that some paging services will reject transactions that have an empty Field 1. data 1 - set selector = TAP Account selector, 1 based. data 2:17 - Pager ID String. This string is up to 16 ASCII characters. Null terminated if fewer than 16 characters are used. The string will be transmitted with escaping as specified by the control-character escaping mask for the given destination.
TAP Service Settings	28	This parameter is used to configure one or more sets of values related to strings, escaping, and timeouts and retries associated with a TAP paging service. The timing parameters are per [TAP], with the exception of T6

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
		and N4, which are extended parameters for this specification. There must be at least one set of TAP Service Setting parameters supported if TAP paging is supported on this channel. data 1 - set selector = TAP Service Setting Selector There is a 1:1 association between the TAP Parameter selector in this row, and the selector in the previous row. Parameter fields that share the same parameter selector form a parameter set. [7:4] - reserved [3:0] - TAP Parameter selector. 1-based. (0 = volatile paramters) data 2 - TAP Confirmation [7:2] - reserved. [1:0] - confirmation. This parameter determines what criteria is used by PEF and the *Alert Immediate* command to determine that a TAP Page was successfully delivered to the paging service. 00b = ACK received after end-of-transaction only 01b = code 211 and ACK received after ETX 10b = code 211or 213, and ACK, received after ETX 11b = reserved data 3:5 - TAP 'SST' Service Type field characters, in ASCII. Default = "PG1". Three characters must be provided. data 6:9 - TAP Control-character escaping mask. (Default = FFFF_FFFFh) [31:0] - each bit position represents escaping for corresponding control characters 31h through 00h. A bit value of 1b = escape the character. 0b = don't escape the character. This bit value is ignored for characters that a required to be escaped by TAP. By default, all control characters are escaped. data 10 - timeout parameters 1 [7:4] TAP T2 - timeout in 500 ms. 0-based (0h = 500 ms). Default = 1h (1 second) [3:0] TAP T1 - timeout in seconds. 0-based (0h = 1 second). Default = 1h (2 seconds) data 11 - timeout parameters 2 [7:4] TAP T4 - timeout in seconds. 0-based (0h = 1 second). Default = 3h (4 seconds) [3:0] TAP T3 - timeout in 2 second increments. 0-based (0h = 2 seconds). Default = 4h (10 seconds) data 12 - timeout parameters 3 [7:4] IPMI T6 - IPMI timeout waiting for end-of-transaction acknowledge, in seconds. 0-based (0 = 1 second). Default = 1h (2 seconds). [3:0] TAP T5 - timeout in 2 second increments. 0-based (0h = 2 seconds). Default = 3h (4 seconds) data 13 - retry parameters 1 [7:4] TAP N2 - retries. 1-based. (0 = no retry). Default = 3. [3:0] TAP N1 - retries. 1-based. (0 = no retry). Default = 3. data 14 - retry parameters 2 [7:4] IPMI N4 - number of retries for end-of-transaction. Default = 3. [3:0] TAP N3 - retries. 1-based. (0 = no retry). Default = 3.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
Terminal Mode Configuration	29	This parameter and its fields only apply when Terminal Mode is enabled. The non-volatile parameters are the initial values used whenever a terminal mode session is first established. The settings are returned to the non-volatile settings when a loss of DCD is detected and whenever the Terminal Mode session is deactivated. data 1 Parameter Operation [7:6] - 00b = Set volatile version of data 1 bits 5:0 and data 2 01b = Set non-volatile version of data 1 bits 5:0 and data 2 10b = Copy non-volatile setting to volatile setting (restore default). 11b = reserved Terminal mode options [5] - 0b = disable line editing 1b = enable line editing [4] - reserved [3:2] - delete control (only applies when line editing is enabled) 00b = BMC outputs a <del> character when <bksp> or <del> is received 01b = BMC outputs a <bksp><sp><bksp> sequence when <bksp> or <del> is received [1] - 0b = no echo 1b = echo (BMC echoes characters it receives) [0] - 0b = disable handshake (See *13.7.7, Terminal Mode Packet Handshake*) 1b = enable handshake data 2 - newline sequences [7:4] - output newline sequence (BMC to console). Selects what characters the BMC uses as a <newline> sequence when the BMC writes a line to the console in Terminal Mode. 0h = no termination sequence 1h = <cr-lf> (default) 2h = <NULL> 3h = <CR> 4h = <LF-CR> 5h = <LF> all other = reserved. [3:0] - input newline sequence (Console to BMC). Selects what characters the console uses as the <newline> sequence when writing to the BMC in Terminal Mode. 0h = reserved 1h = <cr> (default) 2h = <NULL> all other = reserved.
PPP Protocol Options	30	data 1 - Snoop Control [7:3] - reserved [2] - System Negotiation Snooping 1b = BMC snoops system's PPP negotiation (optional) 0b = BMC doesn't snoop system's PPP negotiation [1:0] - Snoop ACCM Control 00b = BMC uses Transmit ACCM when snooping (mandatory if connection mode Auto-detect is supported) 01b = BMC uses Snoop ACCM when snooping (mandatory if connection mode Auto-detect is supported)

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
		10b = reserved 11b = reserved data 2 - Negotiation Control [7:6] - reserved [5:4] - Negotiation Control 00b = BMC Negotiates link parameters (runs LCP) on initial connection and whenever mux becomes switched to BMC and a connection is present. 01b = BMC Negotiates link parameters on initial connection only. Upon a mux switch to the BMC, the BMC continues using the parameters it had originally negotiated. If BMC did not do the negotiation, BMC uses pre-configured settings, following - unless system negotiation snooping is enabled, in which case BMC uses system parameters. 10b = BMC never negotiates link parameters. BMC always uses pre-configured settings unless system negotiation snooping is enabled, in which case BMC uses system parameters. 11b = (optional) [3] - reserved Pre-configured link settings [2] - 1b = BMC uses Transmit ACCM to filter received characters 0b = BMC assumes all control characters 00h-1Fh are escaped [1] - 1b = BMC transmits with Address and Control Field Compression 0b = BMC transmits without Address and Control Field Compression [0] - 1b = BMC transmits with Protocol Field Compression 0b = BMC transmits without Protocol Field Compression data 3 - Negotiation Configuration. This parameter selects what the BMC negotiates for when it runs LCP. [7:5] - reserved [4:3] - BMC PPP IP Address Negotiation. 00b = Request PPP IP Address Assignment. BMC issues an IPCP Configure-Request for IPCP Option 3 “IP Address”. The BMC uses the PPP Account #1’s IP Address parameter (below) as the initial value in the request. If the remote console responds with a different address in a Configure-Nak for option 3, the BMC shall accept that IP Address value and use it as its PPP IP Address. Per [RFC1332], an address of 00.00.00.00 indicates a request to the peer (remote console) to provide the IP Address. If option 3 is rejected, the BMC shall use the PPP Account #1’s IP Address parameter setting for any IP Protocol (0021h) packets it sends to the remote console. The BMC **may** silently discard any IP Protocol packets addressed to an IP Address other than the negotiated PPP IP Address. 01b = Request Fixed PPP IP Address. This is the same as negotiation option 00b “Request PPP IP Address Assignment” except that the BMC will reject any alternative address offered by the remote console, and will continue to request PPP Account #1’s IP Address as the IP Address it will use. 10b = No PPP IP Address Negotiation. The BMC does not issue a Configure-Request to request a PPP IP Address. If this option is selected, the BMC shall accept any IP Protocol (0021h)

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
		message delivered to the Primary or Secondary RMCP Port addresses. The BMC shall use the PPP IP Address parameter setting for any IP Packets it generates. 11b = reserved. [2] - 1b = Enable ACCM negotiation 0b = Disable ACCM negotiation (also use 0b if this option not supported) [1] - 1b = Enable Address and Control Field Compression 0b = Disable Address and Control Field Compression (also use 0b if this option not supported) [0] - 1b = Enable Protocol Field Compression 0b = Disable Protocol Field Compression(also use 0b if this option not supported)
PPP Primary RMCP Port Number (optional)	31	data 1:2 - Primary RMCP Port Number, LS-byte first. Default = 26Fh (RMCP 'Aux Bus Shunt' port)
PPP Secondary RMCP Port Number (optional)	32	data 1:2 - Secondary Port Number, LS-byte first. Default = 298h (RMCP 'Secure Aux Bus' port)

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
PPP Link Authentication	33	data 1 - Link Authentication Type. This configuration option selects whether the PPP Link itself is authenticated or not. Used with IPMI Messaging in PPP Mode, this parameter selects which type of Link Authentication will be used when a remote console initiates the connection and the BMC acts as the 'authenticator'. For PAP, CHAP, and MS-CHAP: The usernames (peer names / peer IDs) and passwords (peer password) used for Link Authentication for IPMI Messaging are obtained from users for which the "Enable User for Link Authentication" bit has been set using the *Set User Access* command. For PAP: The 'peer ID' field in the Authenticate Request from the remote console is expected to hold the username, and the password field the password. The BMC uses the peer ID field contents. Assuming the user is appropriately enabled for the channel, the BMC then compares the stored password with the password that was submitted in the Authenticate Request. For CHAP and MS-CHAP v1 & v2: The remote console responds to a challenge generated by the BMC. The BMC takes the name field from that response and uses it as the username to look up the user and password information from the user configuration information. Assuming the user is appropriately enabled for the channel, the BMC will then use that password to verify the response. If the name field is empty, the BMC attempts to look up the password using the Null username. Note that the BMC also inserts the CHAP Name (parameter 34) in the name field of the challenge it generates. [7:4] - reserved [3:0] -PPP Link Authentication protocol 0h = none 1h = CHAP 2h = PAP [RFC 1334] 3h = MS-CHAP v1 [RFC 2433] BMC requires challenge response to be in Windows NT format. 4h = MS-CHAP v1 [RFC 2433] BMC generates challenge response in LAN Manager format. (LAN Manager format is deprecated in RFC 2433, this option is only provided for implementations that may wish to support connecting to older systems that do not support Windows NT format.) 5h = MS-CHAP v2 [RFC 2759]
CHAP Name (required if CHAP supported)	34	data 1:16 - Null terminated ASCII string for the "system name" used to represent the BMC when it emits a challenge during CHAP. This is only used when dialing in to the BMC. If this parameter is provided, it will also be used by MS-CHAP v1 & v2.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
PPP ACCM (optional)	35	data 1:4 - Receive ACCM, MS-byte first. (ls-bit of ls-byte corresponds to character 00h, ms-bit of ms-byte corresponds to character 1Fh). The BMC uses this field as part of link negotiation. A 1b in a bit position identifies a character the must be escaped in order to be accepted by the BMC. The BMC will ignore any corresponding characters that are not escaped. Note that per [RFC 1662] the BMC is required to accept all escaped characters regardless of whether they're part of the set that the BMC required to be escaped. (If XON/XOFF is used, be sure to include the XON/XOFF characters in the ACCM.) If ACCM Negotiation is not enabled (or this parameter is not supported), the BMC will require that all control characters (00h-1Fh) be escaped. data 5:8 - Transmit ACCM, MS-byte first (ls-bit of ls-byte corresponds to character 00h, ms-bit of ms-byte corresponds to character 1Fh). If ACCM Negotiation is enabled, and this field is supported, this field will determine which characters the BMC will always transmit with escaping. Characters that match the value for the PPP flag character (7Eh) and escape character (7Dh) are always escaped when encountered in the data, so the values in the corresponding bit positions are 'don't care'. I.e. if you set this field to all 0's, the 7Eh and 7Dh will still be escaped before being transmitted. If ACCM Negotiation is enabled, but this field is not supported, the BMC will negotiate to transmit all control characters (00h-1Fh) with escaping. If ACCM Negotiation is not enabled, the BMC will transmit all control characters (00h-1Fh) with escaping.
PPP Snoop ACCM (optional. Required if Connection Mode Auto-detect is supported for PPP mode)	36	data 1:4 - Snoop Receive ACCM, MS-byte first. (ls-bit of ls-byte corresponds to character 00h, ms-bit of ms-byte corresponds to character 1Fh). A 1b in a bit position identifies a character the must be escaped in order to be accepted by the BMC. The BMC can be directed to use this receive ACCM when snooping for a PPP Packet for Connection Mode Auto-detect. This ACCM is used while snooping when the mux is switched over to the system.
Number of PPP Accounts (READ ONLY)	37	data 1 - Number of non-volatile destination IP Addresses for this channel. Account 0 is always present and is typically used as a volatile destination that is used with the *Alert Immediate* command. Account 1 is used for IPMI Messaging via PPP. [7:4] - reserved. [3:0] - 0h = PPP Alerting and Callback are not supported. 9h to Fh = reserved.
PPP Account Dial String Selector	38	data 1 - set selector = account set selector. data 2 - Dial String Selector. Selects which dial string from the Destination Dial Strings to use for calling the given PPP account.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
PPP Account IP Addresses, BMC IP Address	39	This is the IPv4 Address used to connect to a PPP Server for dial-out alerting or callback. This value will be assumed to be the IP Address of the PPP Server unless the PPP Server requests a different address by negotiating IPCP Option 3 (IP Address). The BMC will offer this address to the PPP Server if the PPP Server passes 00.00.00.00 as the requested IP Address when negotiating IPCP option 3. Otherwise, the BMC will accept the IP Address requested by the PPP Server. Account 0 is always present and is typically used as a volatile destination that is used with the *Alert Immediate* command. Account 1 *holds the IP Address used for IPMI Messaging via PPP* (the BMC's IP Address) instead of a PPP Server's IP Address. It is also used as the BMC's IP Address when connecting to a remote system for callback or PPP Alerts. The Account 1 IP Address is handled according to the PPP Protocol Options parameter, above. data 1 - set selector = account set selector. data 2:5 - IP Address. MS-byte first. 0000_0000h = unspecified.
PPP Account User Names	40	This parameter holds the username data for dial-out alerting or callback. For MS-CHAP: The BMC will prefix the PPP Account Domain (parameter 41) to this parameter an use the result in the name field of the response to the challenge. The challenge response is based on the specified algorithm and the PPP Account User Password (parameter #42) for the account. For PAP and CHAP: The BMC uses this parameter to populate the peer ID when generating a PAP authentication request, or in the name field of the response to a CHAP challenge. The challenge is signed using the specified algorithm and the PPP Account User Password (parameter #42) for the account. data 1 - set selector = account set selector. data 2:N - User Name data. ASCII string. 16 characters, max. Null terminated if fewer than 16 characters are used.
PPP Account User Domains	41	Required for dial-out alerting using MS-CHAP v1 or v2. If string is non-empty it will be transmitted as a prefix to the user name. Per [RFC 2433] & [RFC 2759] the domain and user name are separated by a backslash '\' character. This character *is not* automatically added by the BMC and should be entered as the last character of the domain. data 1 - set selector = account set selector. data 2:N - User Domain data. ASCII string. 16 characters, max. Null terminated if fewer than 16 characters are used.
PPP Account User Passwords (Write Only)	42	The PPP Account parameters (selected by the account set selector value) are used for connecting to remote systems for dial-out alerting or callback using PPP/UDP mode. Note, the usernames (peer names) and passwords used for Link Authentication for 'call in' IPMI Messaging are obtained from users for which the "Enable User for Link Authentication" bit has been set using the *Set User Access* command. data 1 - set selector = account set selector. data 2:N - password data. ASCII string. 16 characters max. Null terminated if fewer than 16 characters are used.

Parameter	#	Parameter Data (non-volatile unless otherwise noted)[1]
PPP Account Authentication Settings	43	These parameters are used for 'dial-out' connections. data 1 - set selector = account set selector data 2 - Link Authentication Type [7:4] - reserved [3:0] -PPP Link Authentication protocol. 0h = none (Link Authentication not used) 1h = CHAP 2h = PAP 3h = MS-CHAP v1 [RFC 2433] BMC generates challenge response in Windows NT format 4h = MS-CHAP v1 [RFC 2433] BMC generates challenge response in LAN Manager format. (LAN Manager format is deprecated in RFC 2433, this option is only provided for implementations that may connect and send alerts to older systems) 5h = MS-CHAP v2 [RFC 2759]
PPP Account Connection Hold Times	44	Minimum number of seconds that the call to the given account will be held prior to automatically hanging up the call. Note the connection will only stay open for this time if no other alert or action needs to call a different location or use the channel. Note that an implementation is allowed to terminate the connection on system resets, power on/off transitions, and power cycles. data 1 - set selector = account set selector data 2 - connection hold time in seconds. 1-based.
PPP UDP Proxy IP Header data	45	data 1:4 - Source IP Address. MS-byte first. data 5:8 - Destination IP Address. MS-byte first.
PPP UDP Proxy Transmit Buffer Size (READ ONLY)	46	data 1:2 - Transmit buffer size in bytes. 1-based. This parameter is used to return the size of the PPP UDP Proxy Data transmit buffer. 0000h if PPP UDP Proxy not supported on given channel.
PPP UDP Proxy Receive Buffer Size (READ ONLY)	47	data 1:2 - Receive buffer size in bytes. 1-based. This parameter is used to return the size of the PPP UDP Proxy Data transmit buffer. 0000h if PPP UDP Proxy not supported on given channel.
PPP Remote Console IP Address (optional)[4]	48	data 1:4 - IP Address to offer remote peer if it requests the BMC to provide it an address as part of IPCP Negotiation. MS-byte first.
OEM Parameters	192 : 255	This range is available for special OEM configuration parameters. The OEM is identified according to the Manufacturer ID field returned by the *Get Device ID* command.

1. Choice of system manufacturing defaults is left to the system manufacturer unless otherwise specified.
2. These settings are copied from the corresponding non-volatile values whenever the system is powered up or hard reset.
3. Optional but recommended if [MSVT] is implemented in conjunction with IPMI serial port sharing on the same serial interface.
4. Optional but recommended if PPP supported.
5. Per [MSVT] The BMC should put out an <ESC>* to the remote console after being switched by the <ESC>(sequence and after powering up/waking the system using the <ESC>^ sequence. Refer to [MSVT] for timing requirements.

20.3 Set Serial/Modem Mux Command

This command is used to force or request the selected serial mux to connect the serial connector to the baseboard serial port or the BMC serial port. The command also returns the present setting of the mux.

Table 20-5, Set Serial/Modem Mux Command

	byte	data field
Request Data	1	Channel number. This must correspond to the channel number that the desired serial/modem mux is on. [7:4] - reserved [3:0] - Channel number.
	2	Mux setting <VOLATILE> The BMC can override these settings on power down, power on, and system resets, and change it during system operation when a serial/modem connection is activated or deactivated. Otherwise, the 'mux block' settings are set back to 'allowed' on system power up, power down, power cycles, and resets except when those actions are initiated by the *Chassis Control* command. This enables a remote console to use the 'block' settings to keep connected to the BMC after causing a reset or power state change using the *Chassis Control* command. The BMC power-on default (i.e. when the BMC first gets powered/initialized) is based on the Access Mode setting for the channel (See *Table 13-2, Serial Port Sharing* Access Characteristics). The blocking of 'switch requests' and 'switch forces' only affects the operation of the *Set Serial/Modem Mux* command. Switching caused by other mechanisms such as snooping and changes to system or connection states are not blocked. [7:4] - reserved [3:0] - 0h = get present mux setting/status only 1h = request switch of mux to system 2h = request switch of mux to BMC 3h = force switch of mux to system 4h = force switch of mux to BMC 5h = block requests to switch mux to system 6h = allow requests to switch mux to system 7h = block requests to switch mux to BMC 8h = allow requests to switch mux to BMC
Response Data	1	Completion Code

2	Mux setting. This returns the present state of the mux and the mux change bits from the last *Set Serial/Modem Mux* command. switch request enable settings [7] - 0b = requests to switch mux to system are allowed 1b = requests to switch mux to system are blocked [6] - 0b = requests to switch mux to BMC are allowed 1b = requests to switch mux to BMC are blocked switch status [5:4] - reserved [3] - 0b = no alert presently in progress 1b = alert in progress on channel [2] - 0b = no IPMI or OEM messaging presently active on channel 1b = IPMI or OEM messaging session active on channel [1] - 0b = request was rejected 1b = request was accepted (see note, below) or switch was forced present mux setting [0] - 0b = mux is set to system (system can transmit and receive) 1b = mux is set to BMC (BMC can transmit. System can neither transmit nor receive) Note: Bit 1 will immediately indicate whether the request was accepted. However, if 'mux switch acknowledge' is enabled, it may take seconds before the actual switch occurs. Software that needs to confirm a change of the present mux setting must poll the 'present mux setting' bit until it changes to the new state.

20.4 Get TAP Response Codes Command

This command returns the values for up to the last **five** TAP response codes as an aid to verifying TAP settings. The values are volatile and are not guaranteed to be retained across system or management controllers resets or power on/off changes. The values are automatically cleared to '0', '0', '0' at the start of a TAP page. The command is provided to aid in verifying and debugging the TAP configuration settings.

Table 20-6, Get TAP Response Codes Command

	byte	data field
Request Data	1	Channel number. [7:4] - reserved [3:0] - Channel number.
Response Data	1	Completion Code
	2:4	Most recent (last received) 3-character ASCII response code. MS-char. first.
	5:7	Second to last code.
	8:10	Third.
	11:13	Fourth.
	14:16	Fifth.

20.5 Set PPP UDP Proxy Transmit Data Command

This command is used to load data into the PPP UDP Proxy transmit data buffer. This data is expected to consist of UDP Packet Data starting with the first UDP data byte (byte following UDP checksum) through the last UDP data byte. The BMC fills in the remaining PPP and IP/UDP header information, and takes care of framing and escaping for delivering the data over PPP per [RFC 1662]. The BMC *does not* verify the correctness of the data.

Table 20-7, Set PPP UDP Proxy Transmit Data Command

	byte	data field
Request Data	1	Channel number. [7:4] - reserved [3:0] - Channel number.
	2	Block number. 1-based.
	3:18	Block Data. 16-byte block of packet data to set. Note the management controller does not check to see that the block is filled. All writes start at a 16-byte boundary in the buffer specified by the block number. If fewer than 16-bytes are sent, the BMC will not overwrite any prior data remaining in the block.
Response Data	1	Completion Code

20.6 Get PPP UDP Proxy Transmit Data Command

This command is used to retrieve data that has been written into the PPP UDP Proxy transmit data buffer. The command is primarily to aid in the test and debug of software that uses the PPP UDP Proxy capability.

Table 20-8, Get PPP UDP Proxy Transmit Data Command

	byte	data field
Request Data	1	Channel number. [7:4] - reserved [3:0] - Channel number
	2	Block number to get. 1-based.
Response Data	1	Completion Code
	2:17	Block Data. Note, the BMC always returns 16-bytes of data, even if fewer data bytes were written to the specified block.

20.7 Send PPP UDP Proxy Packet Command

This command is used to initiate the transmission of the PPP UDP Proxy Packet using the data stored in the PPP UDP Proxy transmit data buffer.

Table 20-9, Send PPP UDP Proxy Packet Command

	byte	data field
Request Data	1	Channel number. [7:4] - reserved [3:0] - Channel number.
	2:3	UDP Source Port Number. LS-byte first.
	4:5	UDP Destination Port Number. LS-byte first.
	6:9	Source IP Address. MS-byte first. 00 00 00 00h = Use PPP IP Address associated with this channel. (See *Table 20-4, Serial/Modem Configuration Parameters*)
	10:13	Destination IP Address. MS-byte first. MS-byte first. The *Get Session Info* command can be used to look this up for a given session. Software using the *Send PPP UDP Proxy* command will usually get a Session ID or Session Handle from the Boot Options or from a message retrieved via a *Get Message* command.
	14:15	Number of bytes to send. 1-based.
Response Data	1	Completion Code. Generic, plus the following command specific. 80h = PPP Link is not up 81h = IP Protocol is not up

20.8 Get PPP UDP Proxy Receive Data Command

This command is used to retrieve data from the PPP UDP Proxy receive data buffer. The data buffer holds the complete received PPP IP Packet, from the byte following protocol field up to, but excluding, the FCS field. The BMC handles PPP Framing and extracting the encapsulated IP data, including checking the PPP Header information and the FCS, and translating any escaped data in the packet. The BMC does not check the correctness of the encapsulated IP data. The packet is silently discarded if a bad FCS or partial packet is received.

Table 20-10, Get PPP UDP Proxy Receive Data Command

	byte	data field
Request Data	1	Channel number. [7:4] - reserved [3:0] - Channel number.
	2	[7] - Clear Buffer 1b = clear buffer after returning response to this command. 0b = don't clear buffer after completing this command. [6:0] - Block Number. 1-based. 000_0000b = Get received data length.
Response Data	1	Completion Code. 80h = No packet data available. (Returned when a non-zero block number is used but there's no packet data available.)
		If block number = 000_0000b:
	2:3	Number of received data bytes. 0000h after buffer is emptied until a full packet received. This value can be polled to see when a new packet is available. Software must explicitly clear the buffer after completing the read of each packet. Received packets are volatile. The controller may discard packets on controller resets, system resets, system or controller power on/off changes, the enable/disable of the associated channel, or the enable/disable of PPP mode on the associated channel, on changes to the link up/down state, or changes to the IP protocol up/down state.
		If block number non-zero:
	2:17	Block Data. Note, the BMC implementation is allowed to always return a return a full 16-byte block of data, even if fewer bytes were received in the last block.

20.9 Serial/Modem Connection Active (Ping) Command

This command is also referred to as the "Serial/Modem Ping". It is *sent by the BMC* to tell a remote console application whether the system or the BMC is connected to the serial connector before the remote console sends any messages. If Serial Port Sharing is implemented, this command is also sent out before a mux switch from BMC to the system occurs, and immediately after a switch from the system to the BMC occurs. Refer to *13.3, Serial/Modem Connection Active (Ping) Message* for details about the operation of this command.

When enabled, the Serial/Modem Connection Active message is sent out at a nominal rate of ***once every two seconds, +/- 10%*** *for Basic Mode and Terminal Mode.* The BMC is required to send it's first *Serial/Modem Connection Active* message out within **100 milliseconds** of the serial connection to the BMC being established. For PPP Mode, the *Serial/Modem Connection Active* message will only be sent out before a mux switch from BMC to the system occurs, and immediately after a switch from the system to the BMC occurs,

When the BMC issues the Serial/Modem Connection Active command, it will typically be addressed to remote console software. *Thus, for IPMI serial/modem and LAN connections the responder's address byte should be set to* ***81h****, which is the software ID (SWID) for remote console software.* See *Section 5.5,Software IDs* (SWIDs), for more information.

Table 20-11, Serial/Modem Connection Active Command

	byte	data field
Request Data	1	Session state [7:4] - reserved [3:0] - session state 0h - No session active (password required) 1h - Session active (sent after mux switch to BMC or <ESC>([if enabled] detected - and then periodically afterward) 2h - Switching mux to system
	2	IPMI Version in hexadecimal, LSN first. **51h** corresponds to IPMI 1.5.
Response Data	1	Completion Code

20.10 Callback Command

This command is used to initiate a callback to the selected destination. An error completion code will be returned if the specified destination has not been configured to be a callback destination for the selected channel. This callback is accomplished using IPMI commands. Note that there is also a PPP option to perform callback using CBCP. CBCP callback does not use this command. See *13.6.1, Callback Control Protocol (CBCP) Support*.

If the callback command is initiated over the same connection that the callback is to occur over, the BMC will deliver the response to the callback command, and if the Completion Code is 00h (OK) the BMC will terminate the session, hang-up the phone, and initiate the callback.

Table 20-12, Callback Command

	byte	data field
Request Data	1	Channel number. (This value is required to select which configuration parameters are to be used for callback.) [7:4] - reserved [3:0] - Channel number
	2	Destination Selector Selects which alert destination the callback should go to. [7:4] - reserved [3:0] - destination selector. 0 = use volatile destination info. 1-Fh = non-volatile destination.

Response Data	1	Completion Code. Generic codes, plus following command-specific completion codes: 81h = Callback rejected due to alert in progress on this channel. 82h = Callback rejected due to IPMI messaging session active on the callback channel.

20.11 Set User Callback Options Command

This command is used to configure the callback options associated with a specific user. Note that the options are also channel-specific. An implementation can allow three different callback numbers to be offered as part of the callback negotiation.

Table 20-13, Set User Callback Options Command

	byte	data field
Request Data	1	User ID. (00h = reserved. 01h=Set password and enable/disabled User 1) 7:6 - reserved. 5:0 - User ID. 000000b, 000001b = reserved. (User ID 1 is permanently associated with User 1, the null user name).
	2	Channel Number
	3	User callback capabilities [7:2] - reserved [1] - 1b = user enabled for CBCP callback [0] - 1b = user enabled for IPMI callback
	4	CBCP Negotiation Options. Used when user enabled for CBCP callback, and CBCP is globally enabled in the serial/modem configuration parameters. [7:4] - reserved. [3] - 1b = enable callback to one from list of possible numbers. Allow caller to pick one of a set of phone numbers offered by the BMC. [2] - 1b = enable user-specifiable callback number. Allow caller to specify number to be used for callback. [1] - 1b = enable pre-specified number. Allow caller to request that callback occur to a single, pre-specified number for the user. [0] - 1b = enable No Callback. Allow caller to request that callback not be used.
	5	Callback destination 1. This field holds a Destination Selector that picks which Destination Dial String from the serial/modem configuration parameters to use for callback. This selector is used when the 'pre-specified number' option is used. Otherwise, this is the first number in the list when the "caller selects one number from a list of numbers" option is used. FFh = unspecified. Note, if this field is set to FFh, the BMC should reject CBCP negotiation for the 'pre-specified number' option, even if it is enabled in the CBCP Negotiation Options field, above.
	6	Callback destination 2. This is the second number in the list when the "caller selects one number from a list of numbers" option is used. FFh = unspecified. Note, at least one destination must be specified in order for the 'callback to one from a list of numbers' option to be negotiated, even it that option is enabled in the CBCP Negotiation Options field, above.
	7	Callback destination 3. This is the third number in the list when the "caller selects one number from a list of numbers" option is used. FFh = unspecified. Note, at least one destination must be specified in order for the 'callback to one from a list of numbers' option to be negotiated, even it that option is enabled in the CBCP Negotiation Options field, above.
Response Data	1	Completion Code.

20.12 Get User Callback Options Command

This command is used to return the present settings for the User Callback Options.

Table 20-14, Get User Callback Options Command

	byte	data field
Request Data	1	User ID. (00h = reserved. 01h=Set password and enable/disabled User 1) [7:6] - reserved. [5:0] - User ID. 000000b, 000001b = reserved. (User ID 1 is permanently associated with User 1, the null user name).
	2	Channel Number
Response Data	1	Completion Code.
	2	User callback capabilities [7:2] - reserved [1] - 1b = user enabled for CBCP callback [0] - 1b = user enabled for IPMI callback
	3	CBCP Negotiation Options. Used when user enabled for CBCP callback, and CBCP is globally enabled in the serial/modem configuration parameters. [7:4] - reserved. [3] - 1b = callback to one from list of possible numbers enabled [2] - 1b = user-specifiable callback number enabled. [1] - 1b = callback to pre-specified number enabled. [0] - 1b = No Callback enabled. Allow caller to negotiate that callback not be used.
	4	Callback destination 1. This field holds a Destination Selector that picks which Destination Dial String from the serial/modem configuration parameters to use for callback. This selector is used when the 'pre-specified number' option is used. Otherwise, this is the first number in the list when the "caller selects one number from a list of numbers" option is used. FFh = unspecified.
	5	Callback destination 2. This is the second number in the list when the "caller selects one number from a list of numbers" option is used. FFh = unspecified.
	6	Callback destination 3. This is the third number in the list when the "caller selects one number from a list of numbers" option is used. FFh = unspecified.

21. BMC Watchdog Timer Commands

The BMC implements a standardized 'Watchdog Timer' that can be used for a number of system timeout functions by system management software or by the BIOS. Setting a timeout value of '0' allows the selected timeout action to occur immediately. This provides a standardized means for devices on the IPMB, such as Remote Management Cards, to perform emergency recovery actions. Refer to *Appendix G - Command Assignments*

for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 21-1, BMC Watchdog Timer Commands

Command	Section Defined	O/M
Reset Watchdog Timer	21.5	M
Set Watchdog Timer	21.6	M
Get Watchdog Timer	21.7	M

21.1 Watchdog Timer Actions

The following actions are available on expiration of the Watchdog Timer:

- System Reset
- System Power Off
- System Power Cycle
- Pre-timeout Interrupt (OPTIONAL)

The System Reset on timeout, System Power Off on timeout, and System Power Cycle on timeout action selections are mutually exclusive. The watchdog timer is stopped whenever the system is powered-down. A command must be sent to start the timer after the system powers up.

21.2 Watchdog Timer Use Field and Expiration Flags

The watchdog timer provides a 'timer use' field that indicates the current use assigned to the watchdog timer. The watchdog timer provides a corresponding set of 'timer use expiration' flags that are used to track the type of timeout(s) that had occurred.

The timeout use expiration flags retain their state across system resets and power cycles, as long as the BMC remains powered. The flags are normally cleared solely by the 'Set Watchdog Timer' command; with the exception of the "don't log" flag, which is cleared after every system hard reset or timer timeout.

The Timer Use fields indicate:

BIOS FRB2 timeout — An FRB-2 (fault-resilient booting, level 2) timeout has occurred. This indicates that the last system reset or power cycle was due to the system timeout during POST, presumed to be caused by a failure or hang related to the bootstrap processor[4].

BIOS POST timeout — In this mode, the timeout occurred while the watchdog timer was being used by the BIOS for some purpose other than FRB-2 or OS Load Watchdog.

[4] In a multiprocessor system, the bootstrap processor is defined as the processor that, on system power-up or hard reset, is allowed to run and execute system initialization (BIOS POST) while the remaining processors are held in a idle state awaiting startup by the multiprocessing OS.

OS Load timeout	The last reset or power cycle was caused by the timer being used to 'watchdog' the interval from 'boot' to OS up and running. This mode requires system management software, or OS support. BIOS should clear this flag if it starts this timer during POST.
SMS 'OS Watchdog' timeout	This indicates that the timer was being used by System Management Software. During run-time, System Management Software (SMS) starts the timer, then periodically resets it to keep it from expiring. This periodic action serves as a 'heartbeat' that indicates that the OS (or at least the SMS task) is still functioning. If SMS hangs, the timer expires and the BMC generates a system reset. When SMS enables the timer, it should make sure the 'SMS' bit is set to indicate that the timer is being used in its 'OS Watchdog' role.
OEM	Indicates that the timer was being used for an OEM-specific function.

21.2.1 Using the Timer Use field and Expiration flags

The software that sets the Timer Use field is responsible for managing the associated Timer Use Expiration flag. For example, if system management software sets the timer use to 'SMS/OS Watchdog', then that same system management software is responsible for acting on and clearing the associated Timer Use Expiration flag.

In addition, software should *only* interpret or manage the expiration flags for watchdog timer uses that it set. For example, BIOS should not report watchdog timer expirations or clear the expiration flags for non-BIOS uses of the timer. This is to allow the software that did set the Timer Use to see that a matching expiration occurred.

21.3 Watchdog Timer Event Logging

By default, the BMC will automatically log the corresponding sensor-specific watchdog sensor event when a timer expiration occurs. A "don't log" bit is provided to temporarily disable the automatic logging. The "don't log" bit is automatically cleared (logging re-enabled) whenever a timer expiration occurs.

21.4 Pre-timeout Interrupt

The Watchdog Timer offers a 'Pre-timeout Interrupt' option. This option is enabled whenever the 'Interrupt on timeout' option is selected coincident with any of the other Watchdog Timer actions.

If this option is enabled, the BMC generates the selected interrupt a fixed interval before the timer expires. This feature can be used to allow an interrupt handler to intercept the timeout event before it actually occurs.

The default pre-timeout interrupt interval is one (1) second.

The watchdog timeout action and the pre-timeout interrupt functions are individually enabled. Thus, the Watchdog Timer can be configured so that when it times out it provides just an interrupt, just the selected action, both an interrupt and selected action, or none.

If the pre-timeout interval is set to zero, the pre-timeout action occurs concurrently with the timeout action. Note that if a power or reset action is selected with a pre-timeout interval of zero there is no guarantee that a pre-timeout interrupt handler would have time to execute, or to run to completion.

21.4.1 Pre-timeout Interrupt Support Detection

An application that wishes to use a particular pre-timeout interrupt can check for its support by issuing a *Set Watchdog Timer* command with the desired pre-timeout interrupt selection. If the controller does not return an error completion code, then a *Get Watchdog Timer* command should be issued to verify that the interrupt selection was accepted.

While it can be assumed that a controller that accepts a given interrupt selection supports the associated interrupt, it is recommended that, if possible, an application also generate a test interrupt and verify that the interrupt occurs and the handler executes correctly.

21.4.2 BIOS Support for Watchdog Timer

If a system 'Warm Reset' occurs, the watchdog timer may still be running while BIOS executes POST. Therefore, BIOS should take steps to stop or restart the watchdog timer early in POST. Otherwise, the timer may expire later during POST or after the OS has booted.

21.5 Reset Watchdog Timer Command

The *Reset Watchdog Timer* command is used for starting and restarting the Watchdog Timer from the initial countdown value that was specified in the *Set Watchdog Timer* command.

If a pre-timeout interrupt has been configured, the *Reset Watchdog Timer* command will not restart the timer once the pre-timeout interrupt interval has been reached. The only way to stop the timer once it has reached this point is via the *Set Watchdog Timer* command.

Table 21-2, Reset Watchdog Timer Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code

21.6 Set Watchdog Timer Command

The *Set Watchdog Timer* command is used for initializing and configuring the watchdog timer. The command is also used for stopping the timer.

If the timer is already running, the *Set Watchdog Timer* command stops the timer (unless the "don't stop" bit is set) and clears the Watchdog pre-timeout interrupt flag (see *Get Message Flags* command). BMC hard resets, system hard resets, and the Cold Reset command also stop the timer and clear the flag.

Byte 1 is used for selecting the timer use and configuring whether an event will be logged on expiration.

Byte 2 is used for selecting the timeout action and pre-timeout interrupt type.

Byte 3 sets the pre-timeout interval. If the interval is set to zero, the pre-timeout action occurs concurrently with the timeout action.

Byte 4 is used for clearing the Timer Use Expiration flags. A bit set in byte 4 of this command clears the corresponding bit in byte 5 of the *Get Watchdog Timer* command.

Bytes 5 and 6 hold the least significant and most significant bytes, respectively, of the countdown value. The Watchdog Timer decrement is one count/100 ms. The counter expires when the count reaches zero. If the counter is loaded with zero and the Reset Watchdog command is issued to start the timer, the associated timer events occur immediately.

Table 21-3, Set Watchdog Timer Command

	byte	data field
Request Data	1	Timer Use [7] - 1b = don't log [6] - 1b = don't stop timer on *Set Watchdog Timer* command (new for IPMI v1.5) new parameters take effect immediately. If timer is already running, countdown value will get set to given value and countdown will continue from that point. If timer is already stopped, it will remain stopped. If the pre-timeout interrupt bit is set, it will get cleared.[1] 0b = timer stops automatically when *Set Watchdog Timer* command is received [5:3] - reserved [2:0] - timer use (logged on expiration when "don't log" bit = 0b) 000b = reserved 001b = BIOS FRB2 010b = BIOS/POST 011b = OS Load 100b = SMS/OS 101b = OEM 110b -111b = reserved
	2	Timer Actions [7] - reserved [6:4] - pre-timeout interrupt (logged on expiration when "don't log" bit = 0b) 000b = none 001b = SMI 010b = NMI / Diagnostic Interrupt 011b = Messaging Interrupt (this is the same interrupt as allocated to the messaging interface) 100b,111b = reserved [3] - reserved [2:0] - timeout action 000b = no action 001b = Hard Reset 010b = Power Down 011b = Power Cycle 100b,111b = reserved
	3	Pre-timeout interval in seconds. '1' based.
	4	Timer Use Expiration flags clear (0b = leave alone, 1b = clear timer use expiration bit) [7] - reserved [6] - reserved [5] - OEM [4] - SMS/OS [3] - OS Load [2] - BIOS/POST [1] - BIOS FRB2 [0] - reserved
	5	Initial countdown value, lsbyte (100 ms/count)
	6	Initial countdown value, msbyte
Response Data	1	Completion Code

1. Potential race conditions exist with implementations of this option. If the *Set Watchdog Timer* command is sent just before a pre-timeout interrupt or timeout is set to occur, the timeout could occur before the command is executed. To avoid this condition, it is recommended that software set this value no closer than 3 counts before the pre-timeout or timeout value is reached.

21.7 Get Watchdog Timer Command

This command retrieves the current settings and present countdown of the watchdog timer. The Timer Use Expiration flags in byte 5 retain their states across system resets and system power cycles. With the exception of bit 6 in the Timer Use byte, the Timer Use Expiration flags are cleared using the *Set Watchdog Timer* command. They may also become cleared because of a loss of BMC power, firmware update, or other cause of BMC hard reset. Bit 6 of the Timer Use byte is automatically cleared to 0b whenever the timer times out, is stopped when the system is powered down, enters a sleep state, or is reset.

Table 21-4, Get Watchdog Timer Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Timer Use [7] - 1b =don't log [6] - 1b = timer is started (running) 0b = timer is stopped [5:3] - reserved [2:0] - timer use (logged on expiration if don't log bit = 0) 000b = reserved 001b = BIOS FRB2 010b = BIOS/POST 011b = OS Load 100b = SMS/OS 101b = OEM 110b,111b = reserved
	3	Timer Actions [7] - reserved [6:4] - pre-timeout interrupt 000b = none 001b = SMI 010b = NMI / Diagnostic Interrupt 011b = Messaging Interrupt (this would be the same interrupt as allocated to the messaging interface) 100b,111b = reserved [3] - reserved [2:0] - timeout action 000b = no action 001b = Hard Reset 010b = Power Down 011b = Power Cycle 100b,111b = reserved
	4	Pre-timeout interval in seconds. '1' based.
	5	Timer Use Expiration flags (1b = timer expired while associated 'use' was selected.) [7] - reserved [6] - reserved [5] - OEM [4] - SMS/OS [3] - OS Load [2] - BIOS/POST [1] - BIOS FRB2 [0] - reserved
	6	Initial countdown value, lsbyte (100 ms/count)
	7	Initial countdown, msbyte
	8	Present countdown value, lsbyte. The initial countdown value and present countdown values should match immediately after the countdown is initialized via a *Set Watchdog Timer* command and after a *Reset Watchdog Timer* has been executed. Note that internal delays in the BMC may require software to delay up to 100 ms before seeing the countdown value change and be reflected in the *Get Watchdog Timer* command.
	9	Present countdown value, msbyte

22. Chassis Commands

The following chassis commands are specified for IPMI v1.5. These commands are primarily to provide standardized chassis status and control functions for Remote Management Cards and Remote Consoles that access the BMC. They can also be used for 'emergency' management control functions by system management software. Refer to *Appendix G - Command Assignments*

for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 22-1, Chassis Commands

Command	Section Defined	O/M
Get Chassis Capabilities	22.1	M
Get Chassis Status	22.2	M[1]
Chassis Control	22.3	M[1]
Chassis Reset	22.4	O
Chassis Identify	22.5	O
Set Chassis Capabilities	22.6	O
Set Power Restore Policy	22.7	O
Get System Restart Cause	22.9	O[2]
Set System Boot Options	22.10	O[2]
Get System Boot Options	22.11	O[2]
Get POH Counter	22.12	O

1. These commands are mandatory for standalone server motherboards that include ACPI-based power control capabilities.
2. Highly recommended. These commands should be supported on host systems that support remote reset and power on/off capabilities, since these commands enable remote coordination of the booting process with BIOS.

22.1 Get Chassis Capabilities Command

The *Get Chassis Capabilities* command returns information about which main chassis management functions are present on the IPMB (or virtual IPMB) and what addresses are used to access those functions. This command is used to find the devices that provide functions such as SEL, SDR, and ICMB Bridging so that they can be accessed via commands delivered via a physical or logical IPMB. Note that the command does not include a channel number for the individual functions, therefore all reported functions must be located on the primary IPMB.

Refer to [ICMB] for additional information.

The Chassis Capabilities information is non-volatile. There is no requirement that the information be configurable. The Chassis Device function in a peripheral chassis may be hardcoded with this information. For example, a system that implements the ICMB as an add-on bridge to a BMC will typically be able to have the well known address for the BMC (20h) hardcoded as the address for the Chassis SDR, SEL, and SM Devices, while the Chassis FRU Info Device address could be set with the chassis devices own address.

An add-in device that serves as a bridge device that could be used in different vendors systems may want to provide a way for this information to be configured. The *Set Chassis Capabilities* command is one option for providing this.

Table 22-2, Get Chassis Capabilities Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Capabilities Flags [7:4] - reserved [3] - 1b = provides power interlock (IPMI 1.5) [2] - 1b = provides Diagnostic Interrupt (FP NMI) (IPMI 1.5) [1] - 1b = Provides "Front Panel Lockout" (this indicates that the chassis has capabilities to lock out external power control and reset button or front panel interfaces and/or detect tampering with those interfaces) [0] - 1b = Chassis provides intrusion (physical security) sensor
	3	Chassis FRU Info Device Address. Note: all IPMB addresses used in this command are have the 7-bit I^2C slave address as the most-significant 7-bits and the least significant bit set to 0b. 00h = unspecified.
	4	Chassis SDR Device Address
	5	Chassis SEL Device Address
	6	Chassis System Management Device Address
	(7)	Chassis Bridge Device Address. Reports location of the ICMB bridge function. If this field is not provided, the address is assumed to be the BMC address (20h). Implementing this field is required when the *Get Chassis Capabilities* command is implemented by a BMC, and whenever the Chassis Bridge function is implemented at an address other than 20h.

22.2 Get Chassis Status Command

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

Table 22-3, Get Chassis Status Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Current Power State [7] - reserved [6:5] - power restore policy[1] 00b = chassis stays powered off after AC/mains returns 01b = after AC returns, power is restored to the state that was in effect when AC/mains was lost 10b = chassis always powers up after AC/mains returns 11b = unknown [4] - power control fault 1b = Controller attempted to turn system power on or off, but system did not enter desired state. [3] - power fault 1b = fault detected in main power subsystem. [2] - 1b = Interlock (chassis is presently shut down because a chassis panel interlock switch is active). (IPMI 1.5) [1] - Power overload 1b = system shutdown because of power overload condition. [0] - Power is on 1b = system power is on 0b = system power is off (soft-off S4/S5 or mechanical off)
	3	Last Power Event [7:5] - reserved [4] - 1b = last 'Power is on' state was entered via IPMI command [3] - 1b = last power down caused by power fault [2] - 1b = last power down caused by a power interlock being activated [1] - 1b = last power down caused by a Power overload [0] - 1b = AC failed
	4	Misc. Chassis State [7:4] - reserved [3] - 1b = Cooling/fan fault detected [2] - 1b = Drive Fault [1] - 1b = Front Panel Lockout active (power off / reset via chassis push-buttons disabled.) [0] - 1b = Chassis intrusion active

1. In some installations, the chassis' main power feed may be DC based. For example, -48V. In this case, the power restore policy for AC/mains refers to the loss and restoration of the DC main power feed.

22.3 Chassis Control Command

The following command provides a mechanism for providing power up, power down, and reset control.

Table 22-4, Chassis Control Command

	byte	data field
Request Data	1	[7:4] - reserved [3:0] - chassis control 0h = power down. Force system into soft off (S4/S45) state. This is for 'emergency' management power down actions. The command does not initiate a clean shut-down of the operating system prior to powering down the system. 1h = power up. 2h = power cycle (optional). This command provides a power off interval of at least 1 second following the deassertion of the system's POWERGOOD status from the main power subsystem. It is recommended that no action occur if system power is off (S4/S5) when this action is selected, and that a D5h "Request parameter(s) not supported in present state." error completion code be returned. Note that some implementations may cause a system power up if a power cycle operation is selected when system power is down. For consistency of operation, it is recommended that system management software first check the system power state before issuing a power cycle, and only issue the command if system power is ON or in a lower sleep state than S4/S5. 3h = hard reset. In some implementations, the BMC may not know whether a reset will cause any particular effect and will pulse the system reset signal regardless of power state. If the implementation can tell that no action will occur if a reset is delivered in a given power state, then it is recommended (but still optional) that a D5h "Request parameter(s) not supported in present state." error completion code be returned. 4h = pulse Diagnostic Interrupt. (optional) Pulse a version of a diagnostic interrupt that goes directly to the processor(s). This is typically used to cause the operating system to do a diagnostic dump (OS dependent). The interrupt is commonly an NMI on IA-32 systems and an INIT on Intel® Itanium™ processor based systems. 5h = Initiate a soft-shutdown of OS via ACPI by emulating a fatal overtemperature. (optional) all other = reserved
Response Data	1	Completion Code[1]

1. The implementation is allowed to return the completion code prior to performing the selected control action if necessary.

22.4 Chassis Reset Command

This command was used with early versions of the ICMB. It has been superceded by the *Chassis Control* command and is not recommended for new implementations. Refer to [ICMB] for more information. The *Chassis Reset* command allows chassis logic (excluding the chassis device itself) to be reset. For host systems, this corresponds to a system hard reset.

Table 22-5, Chassis Reset Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code

22.5 Chassis Identify Command

This command causes the chassis to physically identify itself by a mechanism chosen by the system implementation; such as turning on blinking user-visible lights or emitting beeps via a speaker, LCD panel, etc. The *Chassis Identify* command automatically times out and deasserts the indication after a configurable time-out. Software must periodically resend the command to keep the identify condition asserted. This will restart the timeout.

Table 22-6, Chassis Identify Command

	byte	data field
Request Data	(1)[1]	[7:0] - Identify Interval in seconds. 1-based. Timing accuracy = -0/+20%. This field is optional. If this byte is not provided the default timeout shall be 15 seconds -0/+20%. 00h = Turn off Identify
Response Data	1	Completion Code

1. This parameter byte is optionally present. If not provided, the *Chassis Identify* can be used to turn on the Identify indication for the default timeout interval, but cannot be used to turn the indication off.

22.6 Set Chassis Capabilities Command

This command is used to set the values that will be returned for the *Get Chassis Capabilities* command into non-volatile storage associated with the Chassis Device.

This command is recommended for all add-on bridge applications.

Table 22-7, Set Chassis Capabilities Command

	byte	data field
Request Data	1	Capabilities Flags [7:2] - reserved [1] - 1b = Provides Front Panel Lockout (see 22.1, Get Chassis Capabilities) [0] - 1b = Provides intrusion
	2	Chassis FRU Info Device Address (see 22.1, Get Chassis Capabilities for a description of these addresses, their use, and the field formatting)
	3	Chassis SDR Device Address
	4	Chassis SEL Device Address
	5	Chassis SM Device Address
	(6)	Chassis Bridge Device Address
Response Data	1	Completion Code. Note, this command does not return an error completion code if an attempt is made to change a 'read-only' parameter. Software must check which fields in the response match the value from the request by using the Get Chassis Capabilities command.

22.7 Set Power Restore Policy Command

This command can be used to configure the power restore policy. This configuration parameter is kept in non-volatile storage. The power restore policy determines how the system or chassis behaves when AC power returns after an AC power loss. The *Get Chassis Status* command returns the power restore policy setting.

Table 22-8, Set Power Restore Policy Command

	byte	data field
Request Data	1	[7:3] - reserved [2:0] - power restore policy 011b = no change (just get present policy support) 010b = chassis always powers up after AC/mains is applied or returns 001b = after AC/mains is applied or returns, power is restored to the state that was in effect when AC/mains was removed or lost 000b = chassis always stays powered off after AC/mains is applied, power pushbutton or command required to power on system all other = reserved
Response Data	1	Completion Code. A non-zero completion code should be returned if an attempt is made to set a policy option that is not supported.
	2	power restore policy support (bitfield) [7:3] - reserved [2] - 1b = chassis supports always powering up after AC/mains returns [1] - 1b = chassis supports restoring power to state that was in effect when AC/mains was lost [0] - 1b = chassis supports staying powered off after AC/mains returns

1. In some installations, the chassis' main power feed may be DC based. For example, -48V. In this case, the power restore policy for AC/mains refers to the loss and restoration of the DC main power feed.

22.8 Remote Access Boot control

The BMC allows a remote console application to optionally direct the boot process following a command to reset, power-up, or power-cycle the system. The remote console sets Boot Option flags prior to issuing a command to reset, power up, or power-cycle the system. The system BIOS can then read these flags after the system restarts and perform the requested boot operation. This will typically be used to direct the system to boot to an alternative partition or source in order to perform emergency remote recovery operations.

The Boot Option parameter definitions follow the set of Boot Option parameters defined by the DMTF Pre-OS Working Group.

Implementing Remote Access Boot control is *optional.*

22.9 Get System Restart Cause Command

This command returns information about what action last caused the system to restart. BIOS can use this command in conjunction with the System Boot Options as additional information in determining whether to perform the requested boot operation.

Table 22-9, Get System Restart Cause Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Restart Cause [7:4] - reserved [3:0] - 0h = unknown (system start/restart detected, but cause unknown) [required if this condition exists] 1h = Chassis Control command [required] 2h = reset via pushbutton [optional] 3h = power-up via power pushbutton [optional] 4h = Watchdog expiration (see watchdog flags) [required] 5h = OEM [optional] 6h = automatic power-up on AC being applied due to 'always restore' power restore policy (see *22.7, Set Power Restore Policy* Command) [optional] 7h = automatic power-up on AC being applied due to 'restore previous power state' power restore policy (see *22.7, Set Power Restore Policy* Command) [optional] 8h = reset via PEF [required if PEF reset supported] 9h = power-cycle via PEF [required if PEF power-cycle supported]Ah = soft reset (e.g. CTRL-ALT-DEL) [optional] all other = reserved
	3	Channel number. (Channel that command was received over)

22.10 Set System Boot Options Command

This command is used to set parameters that direct the system boot following a system power up or reset. The boot flags only apply for one system restart. It is the responsibility of the system BIOS to read these settings from the BMC and then clear the boot flags.

It is possible that a remote console application could set the boot option flags and then be terminated either accidentally or intentionally. In this circumstance, it's possible that a user initiated system restart could occur hours or even days later. If the boot options were used without examining the reset cause, this could cause an unexpected boot sequence. Thus, the BMC will automatically clear a 'boot flags valid bit' if a system restart is not initiated by a *Chassis Control* command within **60 seconds +/- 10%** of the valid flag being set. The BMC will also clear the bit on any system resets or power-cycles that are not triggered by a *System Control* command. This default behavior can be temporarily overridden using the 'BMC boot flag valid bit clearing' parameter.

Table 22-10, Set System Boot Options Command

	byte	data field
Request Data	1	Parameter valid [7] - 1b = mark parameter invalid / locked 0b = mark parameter valid / unlocked [6:0] - boot option parameter selector
	(2:N)	Boot option parameter data, per *Table 22-12, Boot Option Parameters.* Passing 0-bytes of parameter data allows the parameter valid bit to be changed without affecting the present parameter setting.
Response Data	1	Completion Code. Generic plus the following command-specific completion codes: 80h = parameter not supported. 81h = attempt to set the 'set in progress' value (in parameter #0) when not in the 'set complete' state. (This completion code provides a way to recognize that another party has already 'claimed' the parameters) 82h = attempt to write read-only parameter

22.11 Get System Boot Options Command

This command is used to retrieve the boot options set by the *Set System Boot Options* command.

Table 22-11, Get System Boot Options Command

	byte	data field
Request Data	1	Parameter selector [7] - reserved [6:0] - boot option parameter selector
	2	[7:0] - Set Selector Selects a particular block or set of parameters under the given parameter selector. Write as 00h if parameter doesn't use a Set Selector
	3	[7:0] - Block Selector Selects a particular block *within a set* of parameters. Write as 00h if parameter doesn't use a Block Selector. Note: As of this writing, there are no IPMI-specified Boot Options parameters that use the block selector. However, this field is provided for consistency with other configuration commands and as a placeholder for future extension of the IPMI specification.
Response Data	1	Completion Code. Generic plus the following command-specific completion codes: 80h = parameter not supported.
	2	[7:4] - reserved [3:0] - parameter version. 1h for this specification unless otherwise specified.
	3	Parameter valid [7] - 1b = parameter marked invalid / locked 0b = parameter marked valid / unlocked [6:0] - boot option parameter selector
	4:N	Configuration parameter data, per *Table 22-12, Boot Option Parameters.* If the rollback feature is implemented, the BMC makes a copy of the existing parameters when the 'set in progress' state becomes asserted (See the Set In Progress parameter #0). While the 'set in progress' state is active, the BMC will return data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the BMC returns parameter data from non-volatile storage.

Table 22-12, Boot Option Parameters

Parameter	#	Parameter Data (non-volatile unless otherwise noted)
Set In Progress (volatile)	0	data 1 - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software that some other software or utility is in the process of making changes to the data. An implementation can also elect to provide a 'rollback' feature that uses this information to decide whether to 'roll back' to the previous configuration information, or to accept the configuration change. If used, the roll back shall restore all parameters to their previous state. Otherwise, the change shall take effect when the write occurs. [7:2] - reserved [1:0] - 00b = set complete. If a system reset or transition to powered down state occurs while 'set in progress' is active, the BMC will go to the 'set complete' state. If rollback is implemented, going directly to 'set complete' without first doing a 'commit write' will cause any pending write data to be discarded. 01b = set in progress. This flag indicates that some utility or other software is presently doing writes to parameter data. It is a notification flag only, it is not a resource lock. The BMC does not provide any interlock mechanism that would prevent other software from writing parameter data while. 10b = commit write (optional). This is only used if a rollback is implemented. The BMC will save the data that has been written since the last time the 'set in progress' and then go to the 'set in progress' state. An error completion code will be returned if this option is not supported. 11b = reserved
service partition selector (semi-volatile) [1]	1	data 1 [7:0] - service partition selector. This value is used to select which service partition BIOS should boot using. This document doesn't specify which value corresponds to a particular service partition. 00h = unspecified.
service partition scan (semi-volatile) [1]	2	data 1 [7:2] - reserved [1] - 1b = Request BIOS to scan for specified service partition. BIOS clears this bit after the requested scan has been performed. [0] - 1b = Service Partition discovered. BIOS sets this bit to indicate it has discovered the specified service partition. BIOS must clear this bit on all system resets and power ups, except when a scan is requested.
BMC boot flag valid bit clearing (semi-volatile) [1]	3	data 1- BMC boot flag valid bit clearing. Default = 0000b. [7:5] - reserved [4] - 1b = don't clear valid bit on reset/power cycle caused by PEF [3] - 1b = don't automatically clear boot flag valid bit if *Chassis Control* command not received within 60-second timeout (countdown restarts when a *Chassis Control* command is received) [2] - 1b = don't clear valid bit on reset/power cycle caused by watchdog timeout [1] - 1b = don't clear valid bit on pushbutton reset / soft-reset (e.g. "Ctrl-Alt-Del") [0] - 1b = don't clear valid bit on power up via power pushbutton or wake event

boot info acknowledge (semi-volatile) [1]	4	These flags are used to allow individual parties to track whether they've already seen and handled the boot information. Applications that deal with boot information should check the boot info and clear their corresponding bit after consuming the boot options data. data 1: Write Mask ('write-only'. This field is returned as 00h when read. This is to eliminate the need for the BMC to provide storage for the Write Mask field.) [7] - 1b = enable write to bit 7 of Data field [6] - 1b = enable write to bit 6 of Data field [5] - 1b = enable write to bit 5 of Data field [4] - 1b = enable write to bit 4 of Data field [3] - 1b = enable write to bit 3 of Data field [2] - 1b = enable write to bit 2 of Data field [1] - 1b = enable write to bit 1 of Data field [0] - 1b = enable write to bit 0 of Data field data 2:Boot Initiator Acknowledge Data The boot initiator should typically write FFh to this parameter prior to initiating the boot. The boot initiator may write 0's if it wants to intentionally direct a given party to ignore the boot info. This field is automatically initialized to 00h when the management controller is first powered up or reset. [7] - reserved. Write as 1b. Ignore on read. [6] - reserved. Write as 1b. Ignore on read. [5] - reserved. Write as 1b. Ignore on read. [4] - 0b = OEM has handled boot info. [3] - 0b = SMS has handled boot info. [2] - 0b = OS / service partition has handled boot info. [1] - 0b = OS Loader has handled boot info. [0] - 0b = BIOS/POST has handled boot info.
boot flags (semi-volatile) [1]	5	data 1 [7] - 1b = boot flags valid. The bit should be set to indicate that valid flag data is present. This bit may be automatically cleared based on the boot flag valid bit clearing parameter, above. [6:0] - reserved BIOS support for the following flags is optional. If a given flag is supported, it must cause the specified function to occur in order for the implementation to be considered to be conformant with this specification. The following parameters represent temporary overrides of the BIOS default settings. BIOS should only use these parameters for the single boot where these flags were set. If the bit is 0b, BIOS should use its default configuration for the given option. data 2 [7] - 1b = CMOS clear [6] - 1b = Lock Keyboard [5:2] - Boot device selector 0000b = No override 0001b = Force PXE 0010b = Force boot from default Hard-drive[2] 0011b = Force boot from default Hard-drive, request Safe Mode[2] 0100b = Force boot from default Diagnostic Partition[2] 0101b = Force boot from default CD/DVD[2] 0110b-1110b = Reserved 1111b = Force boot from Floppy/primary removable media [1] - 1b = Screen Blank [0] - 1b = Lock out Reset buttons data 3 [7] - 1b = Lock out (power off/ sleep request) via Power Button [6:5] - Firmware (BIOS) Verbosity (Directs what appears on POST display) 00b = system default 01b = request quiet display 10b = request verbose display 11b = reserved [4] - 1b = Force progress event traps. When set to 1b, the BMC transmits PET traps for

		BIOS progress events to the LAN or serial/modem destination for the session that set the flag. Since this capability uses PET traps, this bit will be ignored if for connection modes that do not support PET such as Basic Mode and Terminal Mode. [3] - 1b = User password bypass. When set to 1b, the managed client's BIOS boots the system and bypasses any user or boot password that might be set in the system. This option allows a system administrator to, for example, force a system boot via PXE in an unattended manner. It is recommended that *ADMINISTRATOR Privilege Level be required to set this bit* if this feature is supported in a given BIOS. [2] - 1b = Lock Sleep Button. When set to 1b, directs BIOS to disable the sleep button operation for the system, normally until the next boot cycle. Client instrumentation might provide the capability to re-enable the button functionality without rebooting. [1:0] - 00b = console redirection occurs per BIOS configuration setting 01b = suppress (skip) console redirection if enabled 10b = request console redirection be enabled 11b = reserved data 4 [7:4] - reserved [3] - BIOS Shared Mode Override[3] Can be used to request BIOS to temporarily place the channel into Shared access mode. Per the recommendations in *Table 13-2, Serial Port Sharing Access Characteristics,* 'Shared' access would cause the baseboard serial controller to both remain enabled after POST/start of OS boot, while also allowing the BMC to be accessible. This can be useful when booting to an alternative device such as a Diagnostic Partition since it means the partition can use the serial port but that communication with the BMC can remain available if the partition software fails. Note: BIOS should only pay attention this field if when the 'valid' flag is set and the 'BIOS/POST has handled boot info' flag is set. 1b = Request BIOS to temporarily set the access mode for the channel specified in parameter #6 to 'Shared'. This is typically accomplished by sending a 'Set Channel Access' command to set the *volatile* access mode setting in the BMC[4]. 0b = No request to BIOS to change present access mode setting. [2:0] - BIOS Mux Control Override Can be used to request BIOS to force a particular setting of the serial/modem mux at the conclusion of POST / start of OS boot. This override takes precedence over the mux settings for the access mode even if the BIOS Shared Mode Override is set. Note: BIOS should only pay attention this field if when the 'valid' flag is set and BIOS/POST has handled boot info flag is set. 000b = BIOS uses recommended setting of the mux at the end of POST (See *Table 13-2, Serial Port Sharing* Access Characteristics for more info.) 001b = Requests BIOS to force mux to *BMC* at conclusion of POST/start of OS-boot. If honored, this will override the recommended setting of the mux at the end of POST (See *Table 13-2, Serial Port Sharing* Access Characteristics for more info.) 010b = Requests BIOS to force mux to *system* at conclusion of POST/start of OS-boot. If honored, this will override the recommended setting of the mux at the end of POST (See *Table 13-2, Serial Port Sharing* Access Characteristics for more info.) data 5 - reserved

boot initiator info (semi-volatile) [1]	6	Address & Identity information for the party that initiated the boot. The party that initiates the boot writes this parameter and the boot info acknowledge parameter prior to issuing the command that causes the system power up, power cycle, or reset. This data is normally written by the remote console application, not the BMC. boot source data 1- Channel Number. Channel that will deliver the boot command (e.g. chassis control). BIOS and boot software (e.g. service partition or OS loader) can use the *Get Channel Sessions* to find out information about the party that initiated the boot. [7:4] - reserved [3:0] - Channel Number data 2:5 - Session ID. Session ID for session that the boot command will be issued over. This value can be used with the *Get Channel Sessions* command to find out information about the party that initiated the boot. data 6:9 - Boot Info Timestamp. This timestamp is used to help software determine whether the boot information is 'stale' or not. A service partition or OS loader may elect to ignore the boot information if it is older than expected. The boot initiator should load this field with the timestamp value from the *Get SEL Time* command prior to issuing the command that initiates the boot.
boot initiator mailbox (semi-volatile) [1][2]	7	This parameter is used as a 'mailbox' for holding information that directs the operation of the OS loader or service partition software. The data content is specified by the software vendor. Note: Since this information will be retained by the BMC and may be readable by other software entities, care should be taken to avoid using it to carry 'secret' data. data1: Set Selector = block selector Selects which 16-byte info block to access. 0-based. data 2:(17) block data The first three bytes of block #0 are required to be an IANA Enterprise ID Number (least significant byte first) for the company or organization that has specified the loader. Up to 16-bytes per block of information regarding boot initiator, based on protocol and medium. An implementation is required to support at least **80-bytes** (five blocks) of storage for this command. Previous values are overwritten. The BMC does not automatically clear any remaining data bytes if fewer than 16 bytes are written to a given block.
OEM Parameters (optional. Non-volatile or volatile as specified by OEM)	96:127	This range is available for special OEM configuration parameters. The OEM is identified according to the Manufacturer ID field returned by the *Get Device ID* command.

1. The designation 'semi-volatile' means that the parameter will be kept across system power cycles, resets, system power on/off, and sleep state changes, but will not be preserved if the management controller loses standby power or is cold reset. Parameters designated as 'semi-volatile' are initialized to 0's upon controller power up or hard reset, unless otherwise specified.
2. IPMI allows software to use the boot initiator mailbox as a way for a remote application to pass OEM parameters for additional selection of the boot process and direction of the startup of post-boot software. If additional parameters are not included, the system boots the primary/first-scanned device of the type specified.
3. When BIOS temporarily changes the access mode to 'Shared', the BMC should operate according to the description for that mode provided in *Table 13-2, Serial Port Sharing Access Characteristics*. Because this is a volatile setting, the BMC will return to operating according to the non-volatile setting on the next system power down or hard reset. A remote application that uses this bit should be aware of possible differences in operation between the non-volatile setting and Shared mode. For example, the differences in answering behavior between "Shared" mode and "Always Available" mode.
4. BIOS should set this access mode and, if serial port sharing is enabled, configure the system UART according to *Table 13-2, Serial Port Sharing Access Characteristics* prior to launching the load (boot) of the operating system. It is recommended that this operation be performed as early in POST as feasible. In any case, a remote application should be aware that the BIOS may be operating according to the non-volatile setting during a significant portion of POST until it reaches the point where it acts on the BOOT options.

22.12 Get POH Counter Command

This version of IPMI provides a specification for an optional, POH (Power-On Hours) counter. The management controller automatically increments non-volatile storage at the specified rate whenever the system is powered up. It is recommended that this command be implemented in the BMC to provide a standardized location for this function.

Note that in a power-managed system, the definition of 'powered up' can be somewhat ambiguous. The definition used here is that the power-on hours shall accumulate whenever the system is in the operational (S0) state. An implementation may elect to increment power-on hours in the S1 and S2 states as well.

'Clear' or 'Set' commands are not specified for this counter. This is because the counter is most typically used for warranty tracking or replacement purposes where changing or clearing the counter would defeat the purpose.

The following command is used for accessing the POH Counter. This command returns the present reading of the counter, plus the number of counts per hour.

Table 22-13, Get POH Counter Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Minutes per count.
	3:6	Counter reading. LS Byte first.

When the system is powered down between counts, the counter either picks up incrementing at the offset at which the power down occurred, or starts counting at 0 minutes from the last counter reading, depending on the choice of the implementer. In any case, the time does not get 'rounded up' to the next count as a result of powering down between counts.

23. Event Commands

The 'Sensor/Event' Network Function is used for device functionality related to the transmission, reception, and handling of 'Event Messages' and platform sensors.

What is commonly referred to as an 'Event Message' is actually a Sensor/Event Message with a command byte of '02h'. The request is also referred to as an 'Event Request Message', while the corresponding response is referred to as an 'Event Response Message'.

The following presents the list of the Event commands under the 'Sensor/Event' Network Function. Refer to *Appendix G - Command Assignments* for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 23-1, Event Commands

Command	Section Defined	Mandatory/Optional	
		Event Generator	Event Receiver
Set Event Receiver	23.1	M	O
Get Event Receiver	23.2	M	O
Platform Event (a.k.a. "Event Message")	23.3	M	M

23.1 Set Event Receiver Command

This global command tells a controller where to send Event Messages. The slave address and LUN of the Event Receiver must be provided. A value FFh for the Event Receiver Slave Address disables Event Message generation entirely. This command is only applicable to management controllers that act as IPMB Event Generators.

A device that receives a 'Set Event Receiver' command shall 're-arm' event generation for all its internal sensors. This means internally re-scanning for the event condition, and updating the event status based on the result. This will cause devices that have any pre-existing event conditions to transmit new event messages for those events.

⇒ An *initial update in progress* bit is provided with the *Get Sensor Reading* and *Get Sensor Event Status* commands to help software avoid getting incorrect event status due to a re-arm. For example, suppose a controller only scans for an event condition once every four seconds. Software that accessed the event status using the *Get Sensor Reading* command could see the wrong status for up to four seconds before the event status would be correctly updated. A controller that has slow updates must implement the initial update in progress bit, and should not generate event messages until the update has completed. Software should ignore the Event Status bits while the *initial update in progress* bit is set.

Table 23-2, Set Event Receiver

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

23.2 Get Event Receiver Command

This global command is used to retrieve the present setting for the Event Receiver Slave Address and LUN. This command is only applicable to management controllers that act as IPMB Event Generators.

Table 23-3, Get Event Receiver Command

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

23.3 Platform Event Message Command

This command may be thought of as a request for the BMC to process the event data that the command contains. Typically, the data will be logged to the System Event Log (SEL). Depending on the implementation, the data may also go to the Event Message Buffer and processed by Platform Event Filtering (PEF).

Table 23-4, Platform Event (Event Message) Command

	IPMB MESSAGING (IPMB, LAN, Serial/Modem, PCI Mgmt. Bus)		SYSTEM INTERFACE	
	byte	data field	byte	data field
Request Data	-	Generator ID (RqSA, RqLUN)	1	Generator ID
	1	EvMRev	2	EvMRev
	2	Sensor Type	3	Sensor Type
	3	Sensor #	4	Sensor #
	4	Event Dir | Event Type	5	Event Dir | Event Type
	5	Event Data 1	6	Event Data 1
	6	Event Data 2	7	Event Data 2
	7	Event Data 3	8	Event Data 3
Response Data	1	Completion Code.	1	Completion Code.

The Generator ID field is a required element of an Event Request Message. For IPMB messages, this field is equated to the Requester's Slave Address and LUN fields. Thus, the Generator ID information is not carried in the data field of an IPMB request message.

For 'system side' interfaces, it is not as useful or appropriate to 'overlay' the Generator ID field with the message source address information, and so it is specified as being carried in the data field of the request.

23.4 Event Request Message Fields

An Event Request Message contains the following fields for the Event Receiver, regardless of whether the message is received from the IPMB or from a 'system side' messaging interface, such as the SMIC. Most of the information is passed in the data field of the message, however, in some cases field information is extracted from the 'message header'.

Table 23-5, Event Request Message Fields

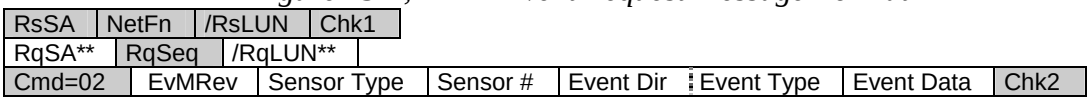
Field	Description
Generator ID	This field identifies the device that has generated the Event Message. This is the 7-bit Requester's Slave Address (RqSA) and 2-bit Requester's LUN (RqLUN) if the message was received from the IPMB, or the 7-bit System Software ID if the message was received from system software.
EvMRev	One byte. Event Message Revision. This field is used to identify different revisions of the Event Message format. *The revision number shall be* **04h** *for Event Messages that comply with the format given in this specification.* IPMI v1.0 messages use 03h. It is recommended that software be able to interpret both versions.
Sensor Type	One byte. Indicates the event class or type of sensor that generated the Event Message. The Sensor Type Codes are specified in *Table 36-3, Sensor Type Codes.*
Sensor #	One byte. A unique number (within a given sensor device) representing the 'sensor' within the management controller that generated the Event Message. Sensor numbers are used for both identification and access of sensor information, such as getting and setting sensor thresholds.
Event Dir	1-bit. Indicates the event transition direction. (0 = Assertion Event, 1 = Deassertion Event)
Event Type	7-bits. This field indicates the type of threshold crossing or state transition (trigger) that produced the event. This is encoded using the Event/Reading Type Code. See *Section 36, Sensor and Event Code Tables.*
Event Data	One to three Bytes. The remainder of the Event Message data according to the class of the Event Type for the sensor (threshold, discrete, or OEM). The contents and format of this field is specified in *Table 23-6, Event Request Message* Event Data Field Contents, below.

The following illustrates which fields from the Event Request Message get transferred to the System Event Record.

23.5 IPMB Event Message Formats

The following figure illustrates the formatting of an Event Request Message as an 'IPMB' message on an I^2C bus, per the *Intelligent Platform Management Bus Communications Protocol v1.0.*

Figure 23-1, IPMB Event Request Message Format



** These fields constitute the 'Generator ID' field for the Event Request Message.
Shading designates fields that are not stored in the event record.

The Event Receiver device responds to IPMB Event Request Messages by simply issuing the Event Response Message with a single 'Completion Code' byte in the data field and a command code of 02h in IPMB Response Message format.

23.6 System Interface Event Request Message Format

Event Request Messages are formatted differently over the System Interface than they are over the IPMB or interfaces that use the IPMB message format. The following figure illustrates the formatting of an Event Request Message as it would be transmitted over the SMIC interface. This is provided for illustration purposes only. Refer

to the individual sections for the System Interfaces for more information: *Section 10.16, Logging Events from System Software via SMIC, Section 9.4, Logging Events from System Software via KCS Interface,* and *Section 11.5, Logging Events from System Software via BT Interface*.

Figure 23-2, Example SMIC Event Request Message Format

NetFn	/00				
Cmd=02	7-bit Software ID**		1		
EvMRev	Sensor Type	Sensor #	Event Dir	Event Type	Event Data

** This field constitutes the 'Generator ID' field for the Event Request Message.
Shading designates fields that are not stored in the event record.

23.7 Event Data Field Formats

The contents of the Event Data field in an Event Request Message (Event Message) is dependent on the sensor class of the sensor. The sensor class obtained from the Event/Reading Type Code specifies whether the sensor event is threshold based, discrete, or OEM defined. Each Event Type is associated with a sensor class. An application can extract the sensor class, and determine the corresponding Event Data format, from the Event/Reading Type Code that was received in the Event Type field in the Event Message. See section *36.1, Event/Reading Type Codes*, for more information.

Table 23-6, Event Request Message Event Data Field Contents

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

	Event Data 2 [7:4] - Optional OEM code bits or offset from 'Severity' Event/Reading Type Code. (0Fh if unspecified). [3:0] - Optional OEM code or offset from Event/Reading Type Code for previous event state. (0Fh if unspecified). Event Data 3 Optional OEM code. FFh or not present or unspecified.

O/M = Optional/Mandatory. *Mandatory* indicates that the byte must be present in all messages. Optional bytes may be left out of messages, as specified. If an optional byte is not present, the Event Receiver shall substitute the value FFh in the corresponding Event Data byte position when transferring the information to the System Event Log function.

24. PEF and Alerting Commands

This section describes the formats of the commands related to configuring and controlling the Platform Event Filtering (PEF) and Alerting capabilities. None of the commands in the following table are required unless PEF or Alerting is supported. Refer to Appendix G - Command Assignments for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 24-1, PEF and Alerting Commands

Command	Section Defined	O/M
Get PEF Capabilities	24.1	M[1]
Arm PEF Postpone Timer	24.2	M[1]
Set PEF Configuration Parameters	24.3	M[1]
Get PEF Configuration Parameters	24.4	M[1]
Set Last Processed Event ID	24.5	M[1]
Get Last Processed Event ID	24.6	M[1]
Alert Immediate	24.7	O[2]
PET Acknowledge	24.8	O[3]

1. Mandatory if PEF or Alerting is supported
2. Mandatory if Alerting is supported
3. Mandatory if LAN or PPP Alerting is supported

24.1 Get PEF Capabilities Command

This command returns the information about the implementation of PEF on the BMC.

Table 24-2, Get PEF Capabilities Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	PEF Version (BCD encoded, LSN first, **51h** for this specification. 51h → version 1.5)
	3	Action Support [7:6] - reserved [5] - 1b = diagnostic interrupt [4] - 1b = OEM action [3] - 1b = power cycle [2] - 1b = reset [1] - 1b = power down [0] - 1b = Alert
	4	Number of event filter table entries (1 based)

24.2 Arm PEF Postpone Timer Command

This command is used by software to enable and arm the PEF Postpone Timer. The command can also be used by software to disable PEF indefinitely during run-time. Once enabled, the timer automatically starts counting down whenever the last software-processed event Record ID is for a record that is not equal to the most recent (last) SEL record. The countdown will begin immediately if the Record IDs are already different when the timer is armed.

In order to keep the PEF Postpone Timer from expiring, software must use the *Set Last Processed Event ID* command to update the last software-processed Record ID to match the value for the last SEL record. This will cause the BMC to stop the timer and rearm it to start counting down from the value that was passed in the *Arm PEF Postpone Timer* command.

The *Get Last Processed Event ID* command can be used to retrieve the present value for the last SEL record's Record ID, the last BMC-processed Record ID, and the last software-processed Record ID.

Table 24-3, Arm PEF Postpone Timer Command

	byte	data field
Request Data	1	[7:0] - PEF Postpone Timeout, in seconds. 01h → 1 second. 00h = disable Postpone Timer (PEF will immediately handle events, if enabled). The BMC automatically disables the timer whenever the system enters a sleep state, is powered down, or reset. 01h - FDh = arm timer. Timer will automatically start counting down from given value when the last-processed event Record ID is not equal to the last received event's Record ID. FEh = Temporary PEF disable. The PEF Postpone timer does not countdown from the value. The BMC automatically re-enables PEF (if enabled in the PEF configuration parameters) and sets the PEF Postpone timeout to 00h whenever the system enters a sleep state, is powered down, or reset. Software can cancel this disable by setting this parameter to 00h or 01h-FDh. FFh = get present countdown value
Response Data	1	Completion Code
	2	Present timer countdown value

24.3 Set PEF Configuration Parameters Command

This command is used for setting parameters such as PEF enable/disable and for entering the configuration of the Event Filter table and the Alert Strings.

Table 24-4, Set PEF Configuration Parameters Command

	byte	data field
Request Data	1	Parameter selector [7] - reserved [6:0] - Parameter selector
	2:N	Configuration parameter data, per *Table 24-6, PEF Configuration Parameters.*
Response Data	1	Completion Code. Generic plus the following command-specific completion codes: 80h = parameter not supported. 81h = attempt to set the 'set in progress' value (in parameter #0) when not in the 'set complete' state. (This completion code provides a way to recognize that another party has already 'claimed' the parameters) 82h = attempt to write read-only parameter

24.4 Get PEF Configuration Parameters Command

This command is used for retrieving the configuration parameters from the *Set PEF Configuration* command.

Table 24-5, Get PEF Configuration Parameters Command

	byte	data field
Request Data	1	[7] - 1b = get parameter revision only. 0b = get parameter [6:0] - Parameter selector
	2	Set Selector (00h if parameter does not require a Set Selector)
	3	Block Selector (00h if parameter does not require a block number)
Response Data	1	Completion Code. Generic plus the following command-specific completion codes: 80h = parameter not supported.
	2	[7:0] - Parameter revision. Format: MSN = present revision. LSN = oldest revision parameter is backward compatible with. **11h** for parameters in this specification.
		The following data bytes are not returned when the 'get parameter revision only' bit is 1b.
	3:N	Configuration parameter data, per *Table 24-6, PEF Configuration Parameters.* If the rollback feature is implemented, the BMC makes a copy of the existing parameters when the 'set in progress' state becomes asserted (See the Set In Progress parameter #0). While the 'set in progress' state is active, the BMC will return data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the BMC returns parameter data from non-volatile storage.

Table 24-6, PEF Configuration Parameters

Parameter	#	Parameter Data
Set In Progress (volatile)	0	data 1 - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software than some other software or utility is in the process of making changes to the data. An implementation can also elect to provide a 'rollback' feature that uses this information to decide whether to 'roll back' to the previous configuration information, or to accept the configuration change. If used, the roll back shall restore all parameters to their previous state. Otherwise, the change shall take effect when the write occurs. [7:2] - reserved [1:0] - 00b = set complete. If a system reset or transition to powered down state occurs while 'set in progress' is active, the BMC will go to the 'set complete' state. If rollback is implemented, going directly to 'set complete' without first doing a 'commit write' will cause any pending write data to be discarded. 01b = set in progress. This flag indicates that some utility or other software is presently doing writes to parameter data. It is a notification flag only, it is not a resource lock. The BMC does not provide any interlock mechanism that would prevent other software from writing parameter data while. 10b = commit write (optional). This is only used if a rollback is implemented. The BMC will save the data that has been written since the last time the 'set in progress' and then go to the 'set in progress' state. An error completion code will be returned if this option is not supported. 11b = reserved

Parameter	#	Parameter Data
PEF control (non-volatile)	1	data 1 [7:4] - reserved [3] - PEF Alert Startup Delay disable. (optional) 1b = enable PEF Alert Startup delay 0b = disable PEF startup delay. [2] - PEF Startup Delay disable. (optional) An implementation that supports this bit should also provide a mechanism that allows the user to Disable PEF in case the filter entries are programmed to cause an 'infinite loop' of PEF actions (such as system resets or power cycles) when the PEF startup delay is disabled. If this bit is not implemented the PEF startup delay must always be enabled. 1b = enable PEF startup delay on manual (pushbutton) system power-ups (from S4/S5) and system resets (including system resets initiated by PEF). 0b = disable PEF startup delay. [1] - 1b = enable event messages for PEF actions. If this bit is set, each action triggered by a filter will generate an event message for the action. These allow the occurrence of PEF-triggered actions to be logged (if event logging is enabled). The events are logged as System Event Sensor 12h, offset 04h. See *Table 36-3, Sensor Type Codes.*) These event messages are also subject to PEF. 0b = disable event messages for PEF actions. [0] - 1b = enable PEF. 0b = disable PEF.
PEF Action global control (non-volatile)	2	data 1 [7:6] - reserved [5] - 1b = enable diagnostic interrupt [4] - 1b = enable OEM action [3] - 1b = enable power cycle action (No effect if power is already off) [2] - 1b = enable reset action [1] - 1b = enable power down action [0] - 1b = enable Alert action
PEF Startup Delay (optional, non-volatile)	3	data 1 - time to delay PEF after a system power-ups (from S4/S5) and resets. Default = 60 seconds. If this parameter is not provided, the default PEF Startup Delay must be implemented. Enable/disable of the delay is configured using the PEF Control parameter, above. If this parameter is supported, a 00h value can also be used to disable the delay if necessary. See *Section 15.4, PEF Startup Delay*, for more information. Note: An implementation that supports this parameter should also provide a mechanism that allows the user to Disable PEF in case the filter entries are programmed to cause an 'infinite loop' of PEF actions under the situation where this parameter is set to too short an interval to allow a user to locally disable PEF. An implementation is allowed to force this parameter to a minimum, non-zero value. PEF Startup Delay [7:0] - PEF Startup Delay in seconds, +/- 10%. 1-based. 00h = no delay.
PEF Alert Startup Delay (optional, non-volatile)	4	data 1 - time to delay Alerts after system power-ups (from S4/S5) and resets. Default = platform-specific. 60-seconds typical, though may be longer on systems that require more startup time before user can take action to disable PEF. If this parameter is not provided, a default PEF Startup Delay, appropriate for the platform, must be implemented. Enable/disable of the delay can also be optionally configured using the PEF Control parameter, above. An implementation can separately implement this parameter and/or the enable/disable bit. PEF Alert Delay [7:0] - PEF Alert Startup Delay in seconds, +/- 10%. 1-based. 00h = no delay.

Parameter	#	Parameter Data
Number of Event Filters (READ ONLY)	5	Number of event filters supported. 1-based. This parameter does not need to be supported if Alerting is not supported. [7] - reserved [6:0] - number of event filter entries. 0 = alerting not supported.
Event Filter Table, (non-volatile)	6	data 1 - Set Selector = filter number. [7] - reserved. [6:0] - Filter number. 1-based. 00h = reserved. data 2:21 - filter data
Event Filter Table Data 1 (non-volatile)	7	This parameter provides an aliased access to the first byte of the event filter data. This is provided to simplify the act of enabling and disabling individual filters by avoiding the need to do a read-modify-write of the entire filter data. data 1 - Set Selector = filter number [7] - reserved [6:0] - Filter number. 1-based. 00h = reserved. data 2 - data byte 1 of event filter data
Number of Alert Policy Entries (READ ONLY)	8	Number of alert policy entries supported. 1-based. This parameter does not need to be supported if Alerting is not supported. [7] - reserved [6:0] - number of alert policy entries. 0 = alerting not supported.
Alert Policy Table (non-volatile)	9	data 1 - Set Selector = entry number [7] - reserved [6:0] - alert policy entry number. 1-based. data 2:4 - entry data
System GUID (non-volatile)	10	data 1 Used to fill in the GUID field in a PET Trap. Stored per *Table 17-10, GUID Format*. [7:1] - reserved [0] -1b = BMC uses following value in PET Trap. 0b = BMC ignores following value and uses value returned from *Get System GUID* command instead. 2:17 - System GUID
Number of Alert Strings (READ ONLY)	11	Number of alert strings supported in addition to Alert String 0. 1-based. This parameter does not need to be supported if Alerting is not supported. [7] - reserved [6:0] - number of alert strings.
Alert String Keys (volatile) & (non-volatile) - see description	12	Sets the keys used to look up Alert String data in PEF. This parameter does not need to be supported if Alerting is not supported. data 1 - Set Selector = Alert string selector. [7] - reserved. [6:0] - string selector. 0 = selects volatile string parameters 01h-7Fh = non-volatile string selectors PEF uses the following Event Filter Number and the Alert String Key fields to look up the string associated with a particular event. String 0 is a special, volatile string reserved for use by the *Alert Immediate* command. The following two fields are used by PEF to look up a particular Alert String based on information obtained from the alert policy entry. The fields should typically be set to 0's (unspecified) for string selector 0. PEF will scan the values for string 0 when doing a look up, so the string 0 values can be set to non-zero values for PEF testing/debug purposes in order to avoid writes to non-volatile storage. data 2 - Event Filter Number [7] - reserved. [6:0] - Filter number. 1-based. 00h = unspecified. data 3 - Alert String Set [7] - reserved [6:0] - Set number for string. 1-based. 00h = unspecified.

Parameter	#	Parameter Data
Alert Strings (volatile) & (non-volatile) - see description.	13	Sets the Alert String data. The string data that should be used is dependent on the Channel and Alert Type. This parameter does not need to be supported if Alerting is not supported. For Dial paging, the BMC automatically follows the string with a <CR> (carriage return) character when sending it to the modem. For TAP paging the string corresponds to 'Field 2', the Pager Message. Note that while the string accepts 8-bit ASCII data, the TAP implementation only supports 7-bit ASCII. The BMC shall automatically zero the 8th bit when transmitting the string during TAP paging. String 0 is a special, volatile string reserved for use by the *Alert Immediate* command. data 1 - Set Selector = string selector. [7] - reserved. [6:0] - string selector. 0 = selects volatile string 01h-7Fh = non-volatile string selectors data 2 - Block Selector = string block number to set, 1 based. Blocks are 16 bytes. data 3:N - String data. Null terminated 8-bit ASCII string. 16-bytes max. per block.
OEM Parameters (optional. Non-volatile or volatile as specified by OEM)	96:127	This range is available for special OEM configuration parameters. The OEM is identified according to the Manufacturer ID field returned by the *Get Device ID* command.

24.5 Set Last Processed Event ID Command

This command is used to set the Record ID for the last event that was processed by system software. For test and debug purposes, it can also be used to set the Record ID for the last event processed by the BMC. See sections *15.3, PEF Postpone Timer* and *15.4.1, Last Processed Event Tracking* for more information. The Last Processed Event ID value is automatically set to FFFFh whenever the SEL is cleared using the *Clear SEL* command. If the *Delete SEL Entry* command is used to either clear the SEL or delete the last event, software must set the Last Processed event manually by using the *Set Last Processed Event ID* command.

Of the two Record IDs (software-processed or BMC-processed) PEF uses the Record ID for the most recent event that was added to the SEL as the indicator of events that have yet to be processed. Both the last BMC-processed and last software-processed IDs are kept in NV storage.

Table 24-7, Set Last Processed Event ID Command

	byte	data field
Request Data	1	[7:1] - reserved. [0] - 0b = set Record ID for last record processed by software. 1b = set Record ID for last record processed by BMC.
	2:3	Record ID. LS-byte first.
Response Data	1	Completion Code 81h = cannot execute command, SEL erase in progress

24.6 Get Last Processed Event ID Command

This command is used to retrieve the Record ID for the last event that was processed by system software and the BMC. See sections *15.3, PEF Postpone Timer* and *15.4.1, Last Processed Event Tracking* for more information.

Table 24-8, Get Last Processed Event ID Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code 81h = cannot execute command, SEL erase in progress
	2:5	Most recent addition timestamp. LS byte first.
	6:7	Record ID for last record in SEL. Returns FFFFh if SEL is empty.
	8:9	Last SW Processed Event Record ID.
	10:11	Last BMC Processed Event Record ID. Returns 0000h when event has been processed but could not be logged because the SEL is full or logging has been disabled.

24.7 Alert Immediate Command

This command is used to send an alert to the destination specified by the destination selector. The kind of alert that will be sent is determined by Destination Type associated with the destination. Alerts that are initiated via this command are never logged as events. This command is to support utilities or BIOS setup options that allow the user to test their alerting configuration for a given destination. The command can also be used by system software as a run-time mechanism to trigger the delivery of an alert.

These alerts are not subject to the Page Blackout intervals, although an alert must complete before the next *Alert Immediate* command will be accepted. *Alert Immediate* commands are also rejected with an error completion code if an IPMI messaging session or automatic page is already in progress.

Table 24-9, Alert Immediate Command

	byte	data field
Request Data	1	Channel number. (This value is required to select which configuration parameters are to be used to send the page.) [7:4] - reserved [3:0] - Channel number.
	2	Destination Selector Selects which alert destination the Alert should go to. [7:4] - reserved [3:0] - destination selector. 0h = use volatile destination info. 1h-Fh = non-volatile destination.
	3	Alert String Selector Selects which Alert String, if any, to use with the alert. [7] - 0b = don't send an Alert String 1b = send Alert String identified by following string selector. [6:0] - string selector. 000_0000b = use volatile Alert String. 01h-7Fh = non-volatile string selector.
Response Data	1	Completion Code. Generic codes, plus following command-specific completion codes:81h = Alert Immediate rejected due to alert already in progress. 82h = Alert Immediate rejected due to IPMI messaging session active on this channel.

24.8 PET Acknowledge Command

This message is used to acknowledge a Platform Event Trap (PET) alert. PET alerts are SNMP Traps that are delivered by LAN or PPP alerting, see [PET] for more info. The PET Acknowledge message is an IPMI Request Message that is sent by the remote console that has received the trap.

Note: The PET Acknowledge command does not require that an IPMI Messaging session be established with the BMC. It is in the same class as the *Get Channel Authentication Capabilities* command. In addition, if Alerting is enabled and the configuration parameters for the Alert Destination require the PET Alert to be acknowledged, the BMC will wait for and accept the PET Acknowledge command until the selected retry interval has expired, even if IPMI Messaging is not available according to the present Access Mode for the channel. For systems using Serial Port Sharing, the BMC will stay switched to the serial connector while waiting for the PET Acknowledge.

Table 24-10, PET Acknowledge Command

	byte	data field
Request Data	1:2	Sequence Number. Value from the Sequence Number field of the PET. LS-byte first[1].
	3:6	Local Timestamp. Value from the Local Timestamp field of the PET. LS-byte first[1].
	7	Event Source type. From corresponding field in the PET.
	8	Sensor Device. From corresponding field in the PET.
	9	Sensor Number. From corresponding field in the PET.
	10:12	Event Data 1:3. From corresponding field in the PET.
Response Data	1	Completion Code.

1. Note: The sequence number and local timestamp fields in the actual PET on the network are in network byte order, therefore filling in these values may require software to re-order the bytes as they get them from the trap.

25. System Event Log (SEL)

The System Event Log is a non-volatile repository for system events and certain system configuration information. The device that fields the commands to access the SEL is referred to as the *System Event Log Device* or *SEL Device.*

Event Message information is normally written into the SEL after being received by the Event Receiver functionality in the Event Receiver Device.

The SEL Device commands are structured in such a way that the SEL Device could actually be separated from the Event Receiver Device. In which case it would be the responsibility of the Event Receiver Device to send the appropriate 'Add SEL Entry' message directly to the SEL Device, or to pass the equivalent request through an intermediary.

SEL Entries have a unique 'Record ID' field. This field is used for retrieving log entries from the SEL. SEL reading can be done in a 'random access' manner. That is, SEL Entries can be read in any order assuming that the Record ID is known.

SEL Record IDs 0000h and FFFFh are reserved for functional use and are not legal ID values. Record IDs are handles. They are *not* required to be sequential or consecutive. Applications should not assume that SEL Record IDs will follow any particular numeric ordering.

SEL Records are kept as an ordered list. That is, appending and deleting individual entries does not change the access order of entries that precede or follow the point of addition or deletion.

25.1 SEL Device Commands

The following table summarizes the commands that are required for implementing a System Event Log device. Note that this specification allows the System Event Log device to be implemented as a separate device from the Event Receiver and Event Generator devices. If this is done, it is up to the implementer to create the method by which Event Messages are passed from the Event Receiver Device to the System Event Log Device. Refer to *Appendix G - Command Assignments* for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 25-1, SEL Device Commands

Command	Section	O/M
Get SEL Info	25.2	M
Get SEL Allocation Info	25.3	O
Reserve SEL	25.4	O[1]
Get SEL Entry	25.5	M
Add SEL Entry	25.6	M[2]
Partial Add SEL Entry	25.7	M[2]
Delete SEL Entry	25.8	O
Clear SEL	25.9	M
Get SEL Time	25.10	M
Set SEL Time	25.11	M
Get Auxiliary Log Status	25.12	O
Set Auxiliary Log Status	25.13	O[3]

1. Mandatory if multiple entities have overlapping access to the SEL. If system mechanisms or conventions are defined that preclude this operation, then this command is optional.
2. Either *Add SEL Entry* or *Partial Add SEL Entry* must be provided. Providing both is optional.

3. *Set Auxiliary Log Status* cannot be implemented without also supporting *Get Auxiliary Log Status.* However, *Get Auxiliary Log Status* is allowed to be implemented without *Set Auxiliary Log Status.*

25.2 Get SEL Info Command

This command returns the number of entries in the SEL, SEL command version, and the timestamp for the most recent entry and delete/clear. The timestamp format is provided in section *31,Timestamp Format.* The *Most Recent Addition* timestamp field returns the timestamp for the last add or log operation, while the *Most Recent Erase* field returns the timestamp for the last delete or clear operation.

These timestamps are independent of timestamps that may be returned by other commands, such as those returned by the Get SDR Repository Info command. The timestamp reflects when the most recent SEL add or erase occurred, *not* when the last add or erase occurred on the physical storage device.

For example, the SEL Info Most Recent Addition timestamp would reflect the last time a new event was added to the SEL. This would be independent of the Most Recent Addition time for an SDR - even if the implementation elected to implement the SEL and SDR Repository in the same storage device.

Table 25-2, Get SEL Info Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code 81h = cannot execute command, SEL erase in progress
	2	SEL Version - version number of the SEL command set for this SEL Device. **51h** for this specification. (BCD encoded).*BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.*
	3	Entries LS Byte - number of log entries in SEL, LS Byte
	4	Entries MS Byte - number of log entries in SEL, MS Byte
	5:6	Free Space in bytes, LS Byte first. FFFFh indicates 65535 or more bytes of free space are available.
	7:10	Most recent addition timestamp. LS byte first. Returns FFFF_FFFFh if no SEL entries have ever been made or if a component update or error caused the retained value to be lost.
	11:14	Most recent erase timestamp. Last time that one or more entries were deleted from the log. LS byte first.
	15	Operation Support [7] - Overflow Flag. 1=Events have been dropped due to lack of space in the SEL. [6:4] - reserved. Write as 000 [3] - 1b = Delete SEL command supported [2] - 1b = Partial Add SEL Entry command supported [1] - 1b = Reserve SEL command supported [0] - 1b = Get SEL Allocation Information command supported

25.3 Get SEL Allocation Info Command

Returns the number of possible allocation units, the amount of usable free space (in allocation units), the allocation unit size (in bytes), and the size of the largest contiguous free region (in allocation units). The 'allocation unit size' is the number of bytes in which storage is allocated. For example, if a 16-byte record is to be added, and the SEL has a 32-byte allocation unit size, then the record would take up 32-bytes of storage.

The SEL implementation shall, at a minimum, support an allocation unit size of ≥16 bytes.

Table 25-3, Get SEL Allocation Info Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Number of possible allocation units, LS Byte
	3	Number of possible allocation units, MS Bytes This number indicates whether the total number of possible allocation units is equal to, or some number less than the log size divided by the allocation unit size. 0000h indicates 'unspecified'.
	4	Allocation unit size in bytes, LS Byte. 0000h indicates 'unspecified'.
	5	Allocation unit size in bytes, MS byte.
	6	Number of free allocation units, LS Byte
	7	Number of free allocation units, MS Byte
	8	Largest free block in allocation units, LS Byte
	9	Largest free block in allocation units, MS Byte
	10	Maximum record size in allocation units.

25.4 Reserve SEL Command

This command is used to set the present 'owner' of the SEL, as identified by the Software ID or by the Requester's Slave Address from the command. The reservation process provides a limited amount of protection on repository access from the IPMB when records are being deleted or incrementally read.

The Reserve SEL command is provided to help prevent deleting the wrong record when doing deletes, to provide a mechanism to avoid clearing the SEL just after a new event has been received, and to prevent receiving incorrect data when doing incremental reads.

The Reserve SEL command does NOT guarantee access to the SEL. That is, the case exists that a pair of requesters could vie for access to the SEL in such a manner that they alternately cancel the reservation that is held by the other - effectively 'deadlocking' each other.

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

The Reservation ID is used in the following manner. Suppose an application wishes to clear the SEL. The application would first 'reserve' the repository by issuing a *Reserve SEL* command. The application would then check that all SEL entries have been handled prior to issuing the *Clear SEL* command.

If an new event had been placed in the SEL after the records were checked, but before the *Clear SEL* command, it is possible for the event to be lost. However, the addition of a new event to the SEL causes the present Reservation ID to be 'canceled'. This would prevent the *Clear SEL* command from executing. If this occurred, the application would repeat the reserve-check-clear process until successful.

Table 25-4, Reserve SEL Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code 81h = cannot execute command, SEL erase in progress
	2	Reservation ID, LS Byte 0000h reserved.
	3	Reservation ID, MS Byte

25.4.1 Reservation Restricted Commands

A Requester must issue a 'Reserve SEL' command prior to issuing any of the following SEL commands. Note that the 'Reserve SEL' command only needs to be reissued if the reservation is canceled. These commands shall be rejected if the Requester's reservation has been canceled.

- Delete SEL Entry command
- Clear SEL command
- Get SEL Entry command (if 'get' is from an offset other than 00h)
- Partial Add SEL Entry command

If the given reservation has been canceled, a 'reservation canceled' completion code shall be returned in the response to the above commands.

Note that the Record ID associated with a given record could change between successive offset 0 'Gets' to that Record ID. For example, the first SEL Entry could change if the SEL were cleared and a new event came in. It is thus the responsibility of the device accessing the SEL to verify that the retrieved record information matches up with the ID information (timestamp, slave address, LUN, sensor ID, etc.) of the event record.

25.4.2 Reservation Cancellation

The SEL Device shall automatically cancel the present SEL reservation after any of the following events occur:

- A SEL entry is added.
- A SEL entry is deleted such that other Record IDs change. As a simplification, an implementation is allowed to cancel the reservation on any SEL entry deletion.
- The SEL is cleared.
- The SEL Device is reset (via hardware or Cold Reset command)
- A new 'Reserve SEL' command is received.

25.5 Get SEL Entry Command

This command is used to retrieve entries from the SEL. The record data field in the response returns the 16 bytes of data from the SEL Event Record.

Table 25-5, Get SEL Entry

	byte	data field
Request Data	1:2	Reservation ID, LS Byte first. Only required for partial Get. Use 0000h otherwise.[1]
	3:4	SEL Record ID, LS Byte first. 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 Return an error completion code if the SEL is empty. 81h = cannot execute command, SEL erase in progress.
	2:3	Next SEL Record ID, LS Byte first (return FFFFh if the record just returned is the last record.) Note: FFFFh is not allowed as the record ID for an actual record. I.e. the Record ID in the Record Data for the last record should not be FFFFh.
	4:N	Record Data, 16 bytes for entire record

1. The reservation ID should be set to 0000h for implementations that don't implement the Reserve SEL command.

25.6 Add SEL Entry Command

This command is provided to enable BIOS to add records to the System Event Log. Normally, the SEL Device and the Event Receiver Device will be incorporated into the same management controller. In this case, BIOS or the system SMI Handler adds its own events to the SEL by formatting an Event Message and transmitting it to the SEL Device, rather than by using this command.

Records are added on after the last record in the SEL. The SEL Device adds the timestamp according to the SEL Record Type (see *25.6.1, SEL Record Type Ranges*, following) when it creates the record. Thus, in some cases the timestamp bytes in the record data are ignored. However, 'dummy' timestamp bytes must still be present in the data.

The record data field that is passed in the request consists of all bytes of the SEL event record. The Record ID field that is passed in the request is just a placeholder. The Record ID field that was passed in the request will be overwritten with a Record ID value that the SEL Device generates before the record is stored. Depending on the Record Type, the entry may also be automatically timestamped (see following section). If the entry is automatically timestamped, the SEL Device will also over-write the four bytes of the record's timestamp field.

Note: The normal mechanism for adding entries to the SEL is via an Event Request message to the Event Receiver device.

Table 25-6, Add SEL Entry

	byte	data field
Request Data	1:16	Record Data, 16 bytes. Refer to section *26, SEL Record Formats*
Response Data	1	Completion Code. Generic, plus following command specific: 80h = operation not supported for this Record Type 81h = cannot execute command, SEL erase in progress
	2:3	Record ID for added record, LS Byte first.

25.6.1 SEL Record Type Ranges

The following lists the ranges used for SEL Record types:

00h - BFh Range reserved for standard SEL Record Types. As of this writing, only type 02h is defined. Records are automatically timestamped unless otherwise indicated.

C0h - DFh Range reserved for timestamped OEM SEL records. These records are automatically timestamped by the SEL Device.

E0h - FFh Range reserved for non-timestamped OEM SEL records. The SEL Device does not automatically timestamp these records. The four bytes passed in the byte locations for the timestamp will be directly entered into the SEL.

25.7 Partial Add SEL Entry Command

This command is a version of the *Add SEL Entry* command that allows the record to be incrementally added to the SEL. The *Partial Add SEL Entry* command must be preceded by a *Reserve SEL* command. The first partial add must be to offset 0000h, and subsequent partial adds must be done sequentially, with no gaps or overlap between the adds.

The add must be completed before any of its contents can be retrieved from the SEL. If the reservation is canceled before the add is completed, the information is discarded and the add must be redone starting at offset 0000h.

When the Record Type directs the BMC to automatically timestamp the record, the BMC will set the timestamp when the last record is transferred.

Note: The normal mechanism for adding entries to the SEL is via an Event Request message to the Event Receiver device.

Table 25-7, Partial Add SEL Entry Command

	byte	data field
Request Data	1:2	Reservation ID, LS Byte first. Only required for partial add. Use 0000h for Reservation ID otherwise.[1]
	3:4	Record ID, LS Byte first. Used when continuing a partial add (non-zero offset into record). Use 0000h for Record ID otherwise.
	5	Offset into record.
	6	In progress. [7:4] - reserved [3:0] - in progress 0h = partial add in progress. 1h = last record data being transferred with this request
	7:N	SEL Record Data
Response Data	1	Completion Code 80h = Record rejected due to mismatch between record length in header data and number of bytes written. (Verifying the length is an *optional* operation for the management controller) 81h = cannot execute command, SEL erase in progress
	2:3	Record ID for added record, LS Byte first.

1. The reservation ID should be set to 0000h for implementations that don't implement the Reserve SEL command.

25.8 Delete SEL Entry Command

Table 25-8, Delete SEL Entry

	byte	data field
Request Data	1:2	Reservation ID, LS Byte first.[1]
	3:4	SEL Record ID to delete, LS Byte first. 0000h = FIRST ENTRY FFFFh = LAST ENTRY
Response Data	1	Completion Code - Generic plus following command specific: 80h = operation not supported for this Record Type 81h = cannot execute command, SEL erase in progress
	2:3	Record ID for deleted record, LS Byte first.

1. The reservation ID should be set to 0000h for implementations that don't implement the Reserve SEL command.

25.9 Clear SEL Command

The command 'erases' all contents of the System Event Log. Since this process may take several seconds, based on the type of storage device, the command also provides a means for obtaining the status of the erasure.

Table 25-9, Clear SEL

	byte	data field
Request Data	1:2	Reservation ID, LS Byte first.[1]
	3	'C' (43h)
	4	'L' (4Ch)
	5	'R' (52h)
	6	AAh = initiate erase. 00h = get erasure status.
Response Data	1	Completion Code
	2	Erasure progress. [7:4] - reserved [3:0] - erasure progress 0h = erasure in progress. 1h = erase completed.

1. The reservation ID should be set to 0000h for implementations that don't implement the Reserve SEL command.

25.10 Get SEL Time Command

This command returns the time from the SEL Device. This time is used by the SEL Device for Event Timestamping.

Table 25-10, Get SEL Time Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2:5	Present Timestamp clock reading. LS byte first. *See Section 31, Timestamp Format.*

25.11 Set SEL Time Command

This command initializes the time in the SEL Device. This time is used by the SEL Device for Event Timestamping.

Table 25-11, Set SEL Time Command

	byte	data field
Request Data	1:4	Time in four-byte format. LS byte first. *See Section 31, Timestamp Format.*
Response Data	1	Completion Code

25.12 Get Auxiliary Log Status Command

This command originated primarily to provide a mechanism that would allow remote software to know whether new information has been added to Machine Check Architecture (MCA) Log. that can be provided. The MCA Log is a storage area that can be implemented in Intel® Itanium™-based computer systems and holds information from an MCA Handler running from system firmware.

For systems that lack MCA, the command can be used to return information about similar OEM-specified logs that may hold extended event information for the platform. Since such logs are usually central resources, this command will typically be implemented by a BMC in a host system, or the chassis controller in a managed peripheral chassis.

Table 25-12, Get Auxiliary Log Status Command

	byte	data field
Request Data	1	Log Type [7:4] - reserved [3:0] - Log Type 00h = MCA Log 01h = OEM 1 02h = OEM 2 all other = reserved
Response Data	1	Completion Code. An error completion code will be returned if the given log type is not supported.
		For Log Type = MCA Log :
	2:5	IPMI Timestamp for when last entry was added to MCA Log, per *section 31,Timestamp Format*.
	6:9	32-bit count of number of entries in MCA Log, LSByte first. FFFF_FFFFh = unspecified.
		For Log Type = OEM 1 or OEM 2:
	2:5	IPMI Timestamp for when last entry was added to log, per *section 31,Timestamp Format*.
	6:8	OEM ID = three byte OEM IANA. IANA Enterprise Number for OEM/Organization that specifies the following log status bytes. Least significant byte first.
	9:16	Log status bytes per OEM identified by OEM ID

25.13 Set Auxiliary Log Status Command

This command can be used by system software or firmware to set the status returned by the *Get Auxiliary Log Status* command. Some implementations may elect to implement solely private mechanism for setting this status, in which case this command may not be provided even if the *Get Auxiliary Log Status* is.

Table 25-13, Set Auxiliary Log Status Command

	byte	data field
Request Data	1	Log Type [7:4] - reserved [3:0] - Log Type 00h = MCA Log 01h = OEM 1 02h = OEM 2 all other = reserved
		For Log Type = MCA Log :
	2:5	IPMI Timestamp for when last entry was added to MCA Log, per *section 31,Timestamp Format*.
	6:9	32-bit count of number of entries in MCA Log, LSByte first. FFFF_FFFFh = unspecified.
		For Log Type = OEM 1 or OEM 2:
	2:5	IPMI Timestamp for when last entry was added to log, per *section 31,Timestamp Format*.
	6:8	OEM ID = three byte OEM IANA. IANA Enterprise Number for OEM/Organization that specifies the following log status bytes. Least significant byte first.
	9:16	Log status bytes per OEM identified by OEM ID
Response Data	1	Completion Code. An error completion code will be returned if the given Log Type is not supported.

26. SEL Record Formats

The following sections present the record formats for SEL entries. Note that these are the 'external' specifications for the records. The actual storage format within the SEL Device implementation may be different.

26.1 SEL Event Records

The following table presents the format of SEL Event Records This is the stored information from Event Messages, as described in *23.4, Event Request Message* Fields.

Table 26-1, SEL Event Records

Byte	Field	Description
1 2	Record ID	ID used for SEL Record access. The Record ID values 0000h and FFFFh have special meaning in the Event Access commands and must not be used as Record ID values for stored SEL Event Records.
3	Record Type	[7:0] - Record Type 02h = system event record C0h-DFh = OEM timestamped, bytes 8-16 OEM defined E0h-FFh = OEM non-timestamped, bytes 4-16 OEM defined
4 5 6 7	Timestamp	Time when event was logged. LS byte first.
8 9	Generator ID	RqSA & LUN if event was generated from IPMB. Software ID if event was generated from system software. Byte 1 [7:1] - 7-bit I^2C . Slave Address, or 7-bit system software ID [0] 0b = ID is IPMB Slave Address 1b = system software ID Byte 2 [7:4] - Channel number. Channel that event message was received over. 0h if the event message was received via the system interface, primary IPMB, or internally generated by the BMC. (New for IPMI v1.5. These bits were reserved in IPMI v1.0) [3:2] - reserved. Write as 00b. [1:0] - IPMB device LUN if byte 1 holds Slave Address. 00b otherwise.
10	EvM Rev	Event Message format version (=**04h** for events in this specification, 03h for IPMI v1.0 Event Messages.) *Note: the BMC must accept Platform Event request messages that are in IPMI v1.0 format (EvMRev=03h) and log them as IPMI v1.5 Records by setting the EvMRev field to 04h and setting the Channel Number in the Generator ID field appropriately for the channel that the event was received from.*
11	Sensor Type	Sensor Type Code for sensor that generated the event
12	Sensor #	Number of sensor that generated the event
13	Event Dir | Event Type	Event Dir [7] - 0b = Assertion event. 1b = Deassertion event. Event Type Type of trigger for the event, e.g. critical threshold going high, state asserted, etc. Also indicates *class* of the event. E.g. discrete, threshold, or OEM. The Event Type field is encoded using the Event/Reading Type Code. See section *36.1*, *Event/Reading Type Codes*. [6:0] - Event Type Code
14	Event Data 1	Per *Table 23-6, Event Request Message* Event Data Field Contents
15	Event Data 2	Per *Table 23-6, Event Request Message* Event Data Field Contents
16	Event Data 3	Per *Table 23-6, Event Request Message* Event Data Field Contents

26.2 OEM SEL Record - Type C0h-DFh

C0h - DFh Range reserved for timestamped OEM SEL records. These records are automatically timestamped by the SEL Device. These records are entered via the *Add SEL or Partial Add SEL commands.*

Table 26-2, OEM SEL Record (Type C0h-DFh)

Byte	Field	Description
1 2	Record ID	ID used for SEL Record access. The Record ID values 0000h and FFFFh have special meaning in the event access commands, and are not to be used as Record ID values for stored SEL Event Records.
3	Record Type	[7:0] Record Type C0h-DFh = OEM system event record
4 5 6 7	Timestamp	Time when event was logged (automatically added by SEL device). LS byte first.
8:10	Manufacturer ID	Manufacturer ID (see *Get Device ID* command for definition)
11:16	OEM Defined	OEM Defined. This is defined according to the manufacturer identified by the Manufacturer ID field.

26.3 OEM SEL Record - Type E0h-FFh

E0h - FFh Range reserved for non-timestamped OEM SEL records. The SEL Device does not automatically timestamp these records. The four bytes passed in the byte locations normally used for the timestamp will be directly entered into the SEL. *SEL viewer applications should not interpret byte positions 4:7 in this record as a timestamp*. These records are entered via the *Add SEL or Partial Add SEL* commands.

Table 26-3, OEM SEL Record (Type E0h-FFh)

Byte	Field	Description
1 2	Record ID	ID used for SEL Record access. The Record ID values 0000h and FFFFh have special meaning in the event access commands, and are not to be used as Record ID values for stored SEL Event Records.
3	Record Type	[7:0] - Record Type E0h-FFh = OEM system event record
4:16	OEM	OEM Defined. This is defined by the system integrator.

27. SDR Repository

This section describes the logical SDR Repository Device, and the commands that are used to access the SDR Repository. This section also describes a companion set of functionality, the Internal Sensor Initialization Agent, that is part of a system that implements this platform and sensor instrumentation specification.

The SDR Repository is intended to hold information indicating the set of management controllers, sensors, and FRU Devices that is *expected to be in the system.* Platform management often requires knowledge of what devices are supposed to be there, as opposed to what devices are detected. This is because an undetected device may be unintentionally absent, which in platform management usually constitutes a failure condition.

For example, suppose the baseboard had connectors for five fans, but only the first four were supposed to be populated. The SDRs for the system would report four fan sensors, one for each of the first four connectors. This tells system management software that any fewer than four fans on the designated connectors would be an error condition. Thus, if the system user unintentionally disconnected a fan, system management software would see an error when it tried to get the fan status. Keeping this information also enables features that allow the platform management hardware itself to take automatic actions based on the 'missing' fan.

The SDR records can be used to represent a custom configuration. Using the same example, suppose a system integrator wanted to attach only three fans to the baseboard, and use them on the *last three* connectors. The SDRs could be changed to report only three fans, and indicate they're on the last three connectors. System management software only pays attention to sensors for which SDRs are present, thus by just adding, deleting, or modifying SDRs a system integrator can change the population of sensors within the constraint of the total available sensors built into the hardware. (I.e. you can't 'create' a sensor that doesn't pre-exist in hardware).

SDRs are kept in a single, centralized Sensor Data Record Repository to simplify the ability for out-of-band applications to get information about the platform management subsystem. This eliminates the need for out-of-band applications, which may be over slow transports, to perform discovery actions. It also is a better mechanism to ensure that the information actually represents what's supposed to be in the system, instead of just what was discovered.

27.1 SDR Repository Device

The SDR Repository Device is a logical device that accepts and responds to SDR Repository commands. The SDR Repository Device isolated from most aspects of the data that is in an SDR. The SDR Device manages SDRs but does not interpret them or take action on the record contents. The exceptions to this is a small set of fixed fields that are used to identify the record and the record type. These fields are contained in the *Record Header* area of the Sensor Data Record.

Another important set of fields are those that are identified as the *Record Key* fields. The combined information in these fields uniquely identifies the record *contents*. The Record Key fields are used for record content identification, while the Record ID field is used for record *access.* For example, a given instance of a sensor will always have the same Record Key information. The Record ID field, however can vary with time as records are added to and removed from the SDR Repository.

The present specification only allows one SDR Repository device per system. For host systems that incorporate a BMC, the SDR Repository is implemented via the BMC. For peripheral chassis that use the ICMB, the device holding the SDR Repository is specified by the *Chassis Capabilities* command (refer to the *Intelligent Chassis Management Bus Bridge* specification).

27.2 Modal and Non-modal SDR Repositories

There are two possible SDR Repository implementations: modal and non-modal. A non-modal SDR Repository can be written to at any time. Writing to the SDR does not impact the operation of other commands in the management controller.

A *modal* SDR Repository is only updated when the controller is in an *SDR Repository update mode*. This provision is made to allow SDR information to be kept in non-volatile storage devices that may require lengthy write operations, or interfere with other controller operations when updated. For example, this could allow the SDR Repository to be stored in a FLASH device that also holds a portion of management controller code. A modal SDR Repository implementation would allow the functions associated with that code to be temporarily unavailable during the update process.

An implementation that provides Modal SDR Repository Updates is not required to support non-modal SDR updates. Generic SDR update software should first issue a *Get SDR Repository Info* command to determine which type of update is supported. If the command returns 'unspecified', update software should first try a modal update by issuing an Enter SDR Update Mode command. If that command is accepted, it should perform the update in SDR Update Mode. If the command is not accepted, it should then attempt to perform a non-modal update.

27.2.1 Command Support while in SDR Repository Update Mode

The controller is only required to support a subset of its normal commands while it is in SDR Repository Update Mode. A completion code of D0h must be returned as the response to any commands that are rejected because the controller is in update mode. The list of commands that must be supported after entering SDR Repository Update mode are listed in the following table. Detailed information is provided in following sections.

The update mode commands must be supported via the system interface to the BMC. If the controller provides an IPMB, it is recommended, but not mandatory, that the IPMB must remain active in SDR Repository Update mode.

Table 27-1, Mandatory SDR Update Mode Commands

Command	Section
Get Device ID	17.1
Get SDR[2]	27.12
Add SDR[1][2]	27.13
Partial Add SDR[1][2]	27.14
Clear SDR Repository[2]	27.16
Exit SDR Repository Update Mode[2]	27.20

1. Either Add SDR or Partial Add SDR must be provided. Providing both is optional.
2. These commands are only accepted from the System Interface if SDR Repository Update Mode was entered via the System Interface, or are only accepted from the device that put the controller into SDR Repository Update mode. Other devices that try to issue these commands will receive a completion code indicating that SDR Repository Update is in progress. Reservation is not required for executing these commands in SDR Repository Update mode.

27.3 Populating the SDR Repository

Most systems are fundamentally static with respect to their platform management configuration once the system integrator has put the system together. Thus, the typical model for the SDR Repository is that it is manually updated using a utility or other piece of software if the platform management configuration is changed in the field.

For example, suppose a system could be upgraded to accept a new RAID backplane that had extra fans and temperature sensors. Part of the upgrade process would be to run a utility, supplied by the system integrator, that updated the SDR Repository with the new SDRs.

27.3.1 SDR Repository Updating

An SDR Repository implementation is not required to implement the *Delete SDR* command. This means that random updates of individual records is not supported. In this case, updating the SDR Repository requires reading out the SDR Repository, updating the copy, clearing the SDR Repository, and writing the updated records in. Note that this approach works for all implementations, and helps avoid potential issues with fragmentation of the SDR Repository.

27.4 Discovering Management Controllers and Device SDRs

IPMI includes the capability for allowing system software to discover management controllers. The responsibility of detecting and integrating new devices is left to system software. This is done to avoid placing additional complexity in BMC firmware, and to allow the discovery and integration policy to be more flexible and sophisticated.

A system can be created that allows new management controllers and SDRs to automatically be discovered and integrated into the SDR Repository. The following steps outline this process:

1. System management software uses the Broadcast Get Device ID command to discover all management controllers on the IPMB. It does this by repeatedly issuing the Broadcast Get Device ID, incrementing the second byte in the message to select different management controller slave addresses. The software only needs to go through the slave addresses that are assignable to IPMB devices (refer to the *IPMB Address Allocation* specification.) System management software can go through this process when it initializes, or, preferably, run this as a 'background' process that scans for new devices during run-time.

2. System management software reads the SDRs and gets a list of the known management controllers from the Management Controller Device Locator records. For each discovered device, system management software checks to see if the device is one of the know devices or not. If the device has a corresponding Management Controller Confirmation record, this record can be used to verify that a different type or instance of controller didn't wind up at the address of a previously present controller.

3. For each newly discovered device, system management software would typically prompt the system user for whether the device should be integrated or not. (For 'missing' devices, the system user would be notified of the change). If the device supports Device SDRs, system management software would be able to read the SDRs from the device and write them to the SDR Repository. If the device didn't include Device SDRs, the software would likely prompt the user for update software supplied by the system integrator or device provider. Note that the management controllers now include information such as the manufacturer ID, that can be an aid to creating useful prompts for this kind of information.

27.5 Reading the SDR Repository

An application that retrieves records from the SDR Repository must first read them out sequentially. This is accomplished by using the *Get SDR* command to retrieve the first SDR of the desired type. The response to this command returns the requested record and the Record ID of the next SDR in sequence in the repository. Note that Record IDs are not required to be sequential or consecutive. Applications should not assume that SDR Record IDs will follow any particular numeric ordering.

The application retrieves succeeding records by issuing a *Get SDR* command using the 'next' Record ID that was returned with the response of the previous *Get SDR* command. This is continued until the 'End of Records' ID is encountered.

Once the application has read out the desired records, it can then randomly access the records according to their Record ID. An application that seeks to access records randomly must save a data structure that retains the Record Key information according to Record ID.

Since it is possible for Record IDs to change with time, it is important for applications to first verify that the Record Key information matches up with the retrieved record. If the Record Key information doesn't match, then the Record ID is no longer valid for that Record Key, and the SDR Records must again be accessed sequentially until the record that matches the Record Key is located.

An application can also tell whether records have changed by examining the 'most recent addition' timestamp using the *Get SDR Repository Info* command.

If record information has changed, an application does not need to list out the entire contents of all records. The *Get SDR* command allows a partial read of the SDR. Thus, an application can search for a given *Record Key* by just retrieving that portion of the record.

27.6 Sensor Initialization Agent

The Sensor Initialization Agent is *not* a logical device, but rather a collection of functions and services that are specific to handling SDR information. Unlike the SDR Repository Device, the Sensor Initialization Agent works directly with the content of SDRs, in particular, the Sensor Data Records and Device Locator Records.

The Initialization Agent utilizes the SDR information for sensor and IPMB Device initialization during system startup. The Initialization Agent knows how to interpret Sensor Data Records and is directed by the 'init required' fields to load thresholds to sensors that have the 'threshold initialization required' bit set in their SDR record. Other bits in the record direct the agent to enable sensors/devices that come up with sensors and/or events disabled.

The Initialization Agent Function normally runs whenever the system powers up, and upon system Hard Resets. This ensures that the sensor subsystem and threshold values will be re-initialized in response to 'push-button' hardware resets. It is also recommended that the Initialization Agent function run when the BMC first receives standby power.

Note that in systems that implement power-management, System Management Software may need to take additional steps to restore intermediate settings after the system has 'woken up'.

27.6.1 System Support Requirements for the Initialization Agent

The BMC requires information about when the system has been powered up, hard reset, or warm 'ctrl-alt-del' reset. This information is needed to trigger the Initialization Agent function. The mechanism for accomplishing this is implementation-dependent. Two common ways to provide this information are via hardware signals to the BMC, or via a BMC-specific application command from BIOS. A combination of the two can also be used. For example, a hardware signals could be used to indicate when the system is hard-reset, while a command from BIOS could indicate warm 'ctrl-alt-del' resets.

27.6.2 IPMI and ACPI Interaction

The Initialization Agent restores 'power-on default' threshold values and event enable settings. In order to provide consistent operation, the initialization agent takes the same actions on 'warm' (e.g. ctrl-alt-del) resets.

In a system that has ACPI, the platform management subsystem cannot generally distinguish between power-up from an S5 'OFF' state and power-up from an S4 'Suspend-to-disk' sleep state. When the system wakes from an S4 state, system management software should recognize this condition so that it can restore any 'volatile' settings that it may have gotten reset by the Initialization Agent.

For other sleep states (S1-S3), the management controllers should retain their settings and the Initialization Agent *should not* be run on wake. If a management controller (other than the BMC) gets powered down in S1-S3, that controller is responsible for retaining the last settings that were written to it by system software.

System management software should also be aware of ACPI interaction with the watchdog timer. The watchdog timer does not automatically stop counting down when the system enters an S1-S3 sleep state. If the watchdog timer is being used as an OS Watchdog, system management software should use support in the operating system to schedule a 'wake event' such that system management software can run and reload the timer before it expires. Alternatively, system management software could shut down the timer upon receiving a notification of entry into a sleep state, but that would reduce the value of using a watchdog timer to monitor OS or system software health.

27.6.3 Recommended Initialization Agent Steps

1. Initialize any BMC internal functions that are required by BIOS during POST.
2. Disable the Event Receiver function for events received from any interface but the system interface, or from BMC internal sensors that require initialization. The BMC should accept event messages from BIOS while the initialization agent is running. The implementation may elect to accept BMC internal event messages from sensors that do not require initialization. It is recommended that any events related to the initialization agent operation are logged during the initialization agent process - but they may be collected and logged at its conclusion.
3. Scan the SDR repository for Management Controller Device Locator records. Collect a list of the addresses of management controllers that require initialization. (A field in the Management Controller Device Locator record indicates whether the management controller requires initialization, and if so, whether event messaging should be enabled after the controller has been initialized.) This list should include the BMC itself.
4. For each Management Controller in the list, turn off Event Generation by using the *Set Event Receiver* command to set the Event Receiver. If the Management Controller does not respond to the *Set Event Receiver* command, take it off the list.
   a) Scan the SDR Repository for Type 01h & Type 02h SDRs. For each encountered:
   b) Check the Device Owner ID to see if the sensor belongs to the BMC or one of the other management controllers in the list. If it does not, go on to the next record.
   c) It is possible that a management controller may have other actions that it takes on an event, thus it is important to disable event scanning before setting thresholds and hysteresis. Check the Sensor Capabilities field to see if per-sensor or per-threshold/per-state disable is supported. If it is, then use the *Set Sensor Event Enable* command to disable scanning and event messages per the SDR.
   d) Set the sensor type, sensor thresholds, and hysteresis as directed by the SDR using the *Set Sensor Type*, *Set Sensor Thresholds*, and *Set Sensor Hysteresis* commands.
   e) Use the *Set Sensor Event Enable* command to enable scanning and event generation per the SDR. Go on to next SDR.
5. Enable the BMC Event Receiver function for the IPMB and other interfaces.
6. For each management controller in the list, enable event message generation or leave it disabled (A field in the Management Controller Device Locator record indicates whether event messaging should be enabled after the controller has been initialized.)

27.7 SDR Repository Device Commands

The following sections describe the commands that an SDR Repository Device provides for accessing the SDR Repository.

The commands are designed to simplify the SDR Repository device's implementation by 'pushing back' intelligence to higher-level software where possible. The SDR Repository device is not intended to be a 'database'

engine. Thus, the SDR access commands do *not* include automatic search functions. It is recommended that an application read the SDR Repository into a RAM buffer and work from that copy (keeping track of the SDR Timestamp to check for possible changes to the SDR Repository). The general procedure for reading SDRs from the SDR Repository is described under the *Get SDR* command.

As with Event Messages, it is also the intent that the commands are designed so that the SDR Repository Device is isolated from needing to know the content and format of the SDR records themselves.

Refer to *Appendix G - Command Assignments* for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 27-2, SDR Repository Device Commands

Command	Section	O/M
Get SDR Repository Info	27.9	M
Get SDR Repository Allocation Info	27.10	O
Reserve SDR Repository	27.11	M
Get SDR	27.12	M[5]
Add SDR	27.13	M[1]
Partial Add SDR	27.14	M[1][5]
Delete SDR	27.15	O[5]
Clear SDR Repository	27.16	M[5]
Get SDR Repository Time	27.17	O/M[2]
Set SDR Repository Time	27.18	O/M[2]
Enter SDR Repository Update Mode	27.19	O[3]
Exit SDR Repository Update Mode	27.20	O[3]
Run Initialization Agent	27.21	O[4]

1. Either *Add SDR* or *Partial Add SDR* command must be provided via the system interface. Providing both via the system interface is optional. For the IPMB, the *Add SDR* and *Partial Add SDR* commands are optional.
2. If the SEL Device and SDR Repository Device are implemented in separate controllers, then both these commands are Mandatory for the SDR Repository Device. If the SDR Repository Device shares the same controller as the SEL Device (This is normally indicated in the IPM Device Support field of the Get Device ID command), then the SDR device uses the SEL Device's Timestamp Clock. In this case, the Get SDR Repository Time command is optional, and the Set SDR Repository Time command is *not used*.
3. Support for both these commands is mandatory for a modal SDR Repository. The *Enter SDR Repository Update Mode* command is mandatory when in 'operational' mode, while the *Exit SDR Repository Update Mode* is mandatory when in 'update' mode.
4. Highly recommended. This supports utilities that can update the SDRs during run-time. Without this, a system reset will need to be performed to cause the initialization agent to run.
5. A reservation field of 0000h is passed to these commands when in SDR Repository Update Mode.

27.8 SDR 'Record IDs'

In order to generalize SDR access, Sensor Data Records are accessed using a 'Record ID' number. There are a fixed number of possible Record IDs for a given implementation of the SDR Repository.

The most common implementation of 'Record IDs' is as a value that translates directly to an 'index' or 'offset' into the SDR Repository. However, it is also possible for an implementation to provide a level of indirection, and implement Record IDs as 'handles' to the Sensor Data Records.

Record ID values may be 'recycled'. That is, the Record ID of a previously deleted SDR can be used as the Record ID for a new SDR. The requirement is that, at any given time, the Record IDs are unique for all SDRs in the repository.

Record IDs can be reassigned by the SDR Repository Device as needed when records are added or deleted. An application that uses a Record ID to directly access a record should always verify that the retrieved record

information matches up with the ID information (slave address, LUN, sensor ID, etc.) of the desired sensor. An application that finds that the SDR at a given 'Record ID' has moved will need to re-enumerate the SDRs by listing them out using a series of *Get SDR* commands. Note that it is not necessary to read out the full record data to see if the Record ID for a particular record has changed. Software can determine whether a given record has been given a different Record ID by examining just the SDR's header and record key bytes.

27.9 Get SDR Repository Info Command

This command returns the SDR command version for the SDR Repository. It also returns a timestamp for when the last ADD, DELETE, or CLEAR occurred. The *Most Recent Addition* timestamp field returns the timestamp for the last addition operation, while the *Most Recent Erase* field returns the timestamp for the last delete or clear operation.

These timestamps are independent of timestamps that may be returned by other commands, such as those returned by the Get SEL Info command. The timestamp reflects when the most recent SDR Repository add or erase occurred, *not* when the last add or erase occurred on the physical storage device.

For example, the SDR Repository Info *Most Recent Addition timestamp* would reflect the last time a new record was added to the SDR Repository. The SDR Repository's most recent addition timestamp is always independent of the most recent addition time for the SEL - even if the SEL and SDR Repository are implemented in the same physical storage device.

Table 27-3, Get SDR Repository Info Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	SDR Version - version number of the SDR command set for the SDR Device. **51h** for this specification. *(BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.)*
	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', FFFEh indicates 64KB-2 *or more* available. FFFFh indicates 'unspecified'.
	7:10	Most recent addition timestamp. LS byte first.
	11:14	Most recent erase (delete or clear) timestamp. LS byte first.
	15	Operation Support [7] - Overflow Flag. 1=SDR could not be written due to lack of space in the SDR Repository. [6:5] - 00b = modal/non-modal SDR Repository Update operation unspecified 01b = non-modal SDR Repository Update operation supported 10b = modal SDR Repository Update operation supported 11b = both modal and non-modal SDR Repository Update supported [4] - reserved. Write as 0b [3] - 1b=Delete SDR command supported [2] - 1b=Partial Add SDR command supported [1] - 1b=Reserve SDR Repository command supported [0] - 1b=Get SDR Repository Allocation Information command supported

27.10 Get SDR Repository Allocation Info Command

Returns the number of possible allocation units, the amount of usable free space (in allocation units), the allocation unit size (in bytes), and the size of the largest contiguous free region (in allocation units). The 'allocation unit size' is the number of bytes in which storage is allocated. For example, if a 20-byte record is to be added, and the SDR Repository has a 16-byte allocation unit size, then the record would take up 32-bytes of storage.

The SDR Repository implementation shall, at a minimum, provide an allocation unit size of ≥16 bytes and a "maximum record size" supporting a record of ≥ 64 bytes.

Software should assume an allocation unit size of 16-bytes if this command is not implemented.

Table 27-4, Get SDR Repository Allocation Info Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2 3	Number of possible allocation units, LS Byte Number of possible allocation units, MS Bytes This number indicates whether the total number of possible allocation units is equal to, or some number less than the log size divided by the allocation unit size. 0000h indicates 'unspecified'.
	4 5	Allocation unit size in bytes. 0000h indicates 'unspecified'.
	6 7	Number of free allocation units, LS Byte Number of free allocation units, MS Byte
	8 9	Largest free block in allocation units, LS Byte Largest free block in allocation units, MS Byte
	10	Maximum record size in allocation units.

27.11 Reserve SDR Repository Command

This command is used to set the present 'owner' of the repository, as identified by the 'Software ID' or by the Requester's Slave Address from the command. The reservation process provides a limited amount of protection on repository access from the IPMB when records are being deleted or incrementally read.

The Reserve SDR Repository command is provided to help prevent deleting the wrong record when doing deletes, and to prevent receiving incorrect data when doing incremental reads.

The Reserve SDR Repository command does NOT guarantee access to the SDR Repository. That is, the case exists that a pair of requesters could vie for access to the SDR in such a manner that they alternately cancel the reservation that is held by the other - effectively 'deadlocking' each other.

A 'Reservation ID' value is returned in response to this command. This value is required in other requests, such as the 'Delete SDR' command. These commands will not execute unless the correct Reservation ID value is provided.

The Reservation ID is used in the following manner. Suppose an application wishes to delete a particular record. The application would first 'reserve' the repository by issuing a *Reserve SDR Repository* command. The application would then read the header and key information from the record to verify that it has the correct Record ID for the record. Assuming this is correct, the application would then issue a *Delete SDR* command using the Reservation ID and Record ID as parameters.

If an event had occurred that changed the Record IDs after the header and key information was read but before the *Delete SDR* command, the *Delete SDR* command could be issued with the Record ID for the wrong record. However, events that change Record IDs for any existing records cause the present Reservation ID to be 'canceled'. This prevents software from using an out-of-date Record ID to access a record. For example, it would

prevent the *Delete SDR* command from executing and deleting the wrong record in case a given Record ID was reassigned to a different record.

Table 27-5, Reserve SDR Repository Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Reservation ID, LS Byte
	3	Reservation ID, MS Byte

27.11.1 Reservation Restricted Commands

A Requester must issue a 'Reserve SDR Repository' command prior to issuing any of the following SDR Repository commands. Note that the 'Reserve SDR Repository' command only needs to be reissued if the reservation is canceled. These commands shall be rejected if the Requester's reservation has been canceled.

- Delete SDR command
- Clear SDR Repository command
- Get SDR command (if a partial read)
- Partial Add SDR command

If the given reservation has been canceled, a 'reservation canceled' completion code shall be returned in the response to the above commands. This is explained further in the next section.

Note that since Record IDs could change between offset 0 'Gets' of a given record, it is the responsibility of the device accessing the repository to verify that the retrieved record information matches up with the ID information (slave address, LUN, sensor ID, etc.) of the desired sensor.

27.11.2 Reservation Cancellation

The SDR Repository Device shall automatically cancel the present SDR Repository reservation after any of the following events occur:

- An SDR record is added using the *Add SDR* command such that other Record IDs change. As a simplification, an implementation is allowed to cancel the reservation on *any* SDR record add.
- An SDR record is deleted such that other Record IDs change. As a simplification, an implementation is allowed to cancel the reservation on *any* SDR record deletion.
- The SDR Repository is cleared.
- The SDR Repository Device is reset (via hardware or Cold Reset command)
- A new 'Reserve SDR Repository' command is received.

An error completion code will be returned if an attempt is made to execute a command that requires a reservation ID, but the reservation ID used is not valid or current.

27.12 Get SDR Command

Returns the sensor record specified by 'Record ID'. The command also accepts a 'byte range' specification that allows just a selected portion of the record to be retrieved (incremental read). The Requester must first reserve the SDR Repository using the 'Reserve SDR Repository' command in order for an incremental read to an offset other

than 0000h to be accepted. (It is also recommended that an application use the *Get SDR Repository Info* command to verify the version of the SDR Repository before it sends any other SDR Repository commands. This is important since the SDR Repository command format and operation can change between versions.)

If 'Record ID' is specified as 0000h, this command returns the Record Header for the 'first' SDR in the repository. FFFFh specifies that the 'last' SDR in the repository should be listed. If 'Record ID' is non-zero, the command returns the information from the matching record, and the Record ID for the *next* SDR in the repository.

An application that wishes to retrieve the full set of SDR Records must first issue the Get SDR starting with 0000h as the Record ID to get the first record. The Next Record ID is extracted from the response and this is then used as the Record ID in a Get SDR request to get the next record. This is repeated until the 'Last Record ID' value (FFFFh) is returned in the 'Next Record ID' field of the response.

A partial read from offset 0000h into the record can be used to extract the header and associated 'Key Fields' for the specified Sensor Data Record in the SDR Repository. An application can use the command in this manner to get a list of what records are in the SDR and to identify the instances of each type. It can also be used to search for an particular sensor record.

Note: to support future extensions, applications should check the SDR Version byte prior to interpreting any of the data that follows.

The application issuing 'Get SDR' commands with a non-zero value for the *Offset into record* field must first reserve the SDR Repository by issuing a 'Reserve SDR Repository' command.

If you issue a *Get SDR* command (storage 23h) with a 'bytes to read' size of 'FFh' - meaning 'read entire record'. A value of 'FFh' will cause an error in most cases, since SDRs are bigger than the buffer sizes for the typical system interface implementation. The controller therefore returns an error completion code if the number of record bytes exceeds the maximum transfer length for the interface. The completion code CAh that indicates that the number of requested bytes cannot be returned. Returning this code is recommended, although a controller could also return an 'FFh' completion code. In either case, the algorithm for handling this situation is to "default to using partial reads if the 'read entire record' operation fails" (that is, if you get a non-zero completion code).

Table 27-6, Get SDR Command

	byte	data field
Request Data	1	Reservation ID. LS Byte. Only required for partial reads with a non-zero 'Offset into record' field. Use 0000h for reservation ID otherwise.
	2	Reservation ID. MS Byte.
	3	Record ID of record to Get, LS Byte
	4	Record ID of record to Get, MS Byte
	5	Offset into record
	6	Bytes to read. FFh means read entire record.
Response Data	1	Completion Code
	2	Record ID for next record, LS Byte
	3	Record ID for next record, MS Byte
	4:3+N	Record Data

27.13 Add SDR Command

This command adds the specified sensor record to the SDR Repository and returns its 'Record ID'. The data passed in the request must contain the SDR data in its entirety.

Table 27-7, Add SDR Command

	byte	data field
Request Data	1:N	SDR Data
Response Data	1	Completion Code
	2	Record ID for added record, LS Byte
	3	Record ID for added record, MS Byte

27.14 Partial Add SDR Command

This command is a version of the *Add SDR* command that allows the record to be incrementally added to the repository. The *Partial Add SDR* command must be preceded by a 'Reserve SDR Repository' command. The first partial add must be to offset 0000h, and partial adds must be done sequentially, with no gaps or overlap between the adds.

The add must be completed before any of its contents can be retrieved from the SDR Repository. If the reservation is canceled before the add is completed, the information is discarded and the add must be redone starting at offset 0000h.

Software should assume an allocation unit size of 16-bytes if the *Get SDR Allocation Info* command is not supported.

Table 27-8, Partial Add SDR Command

	byte	data field
Request Data	1	Reservation ID, LS Byte.
	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	Offset into record.
	6	In progress. [7:4] - reserved [3:0] - in progress 0h = partial add in progress. 1h = last record data being transferred with this request
	7:N	SDR Record Data
Response Data	1	Completion Code. Generic, plus following command-specific: 80h = Record rejected due to mismatch between record length in header data and number of bytes written. (Verifying the length is an *optional* operation for the management controller)
	2	Record ID for added record, LS Byte
	3	Record ID for added record, MS Byte

27.15 Delete SDR Command

Deletes the sensor record specified by 'Record ID'. The Requester's ID and the 'Reservation ID' must also match the present 'owner' of the SDR Repository.

Table 27-9, Delete SDR Command

	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
	2	Record ID for deleted record, LS Byte
	3	Record ID for deleted record, MS Byte

27.16 Clear SDR Repository Command

Clears all records from the SDR Repository and reinitializes the SDR Repository 'subsystem'. Mainly a development and production aid, use of this command should be generally avoided in utilities and system management software. The Requester's ID and Reservation ID information must also match the present 'owner' of the SDR Repository.

Table 27-10, Clear SDR Repository Command

	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	AAh = initiate erase. 00h = get erasure status.
Response Data	1	Completion Code
	2	Erasure progress. [7:4] - reserved [3:0] - erasure in progress 0h = erasure in progress. 1h = erase completed.

27.17 Get SDR Repository Time Command

This command returns the time from the SDR Repository Device. This time is used by the SDR Repository Device for tracking when changes to the SDR Repository have been made. The time keeping format is specified in *Section 31, Timestamp Format.*

A device that contains both a logical SEL device and an SDR Repository device can elect to implement just a single Timestamp Clock, in which case, the *Set SDR Repository* command shall not be used. Instead, the *Set SEL Time* command will be used for setting the time, and the *Get SDR Repository Time* and *Get SEL Time* commands shall effectively return the same time values.

Table 27-11, Get SDR Repository Time Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2:5	Time in four-byte format. LS byte first.

27.18 Set SDR Repository Time Command

This command initializes the time in the SDR Repository Device. This time is used by the SDR Device for tracking when SDR Repository changes have been made. The time keeping format is specified in *Section 31,* Timestamp Format.

A device that contains both a logical SEL device and an SDR Repository device can elect to implement just a single Timestamp Clock, in which case, the *Set SDR Repository* command shall not be used. Instead, the *Set SEL Time* command will be used for setting the time, and the *Get SDR Repository Time* and *Get SEL Time* commands shall effectively return the same time values.

Table 27-12, Set SDR Repository Time Command

	byte	data field
Request Data	1:4	Time in four-byte format. LS byte first.
Response Data	1	Completion Code

27.19 Enter SDR Repository Update Mode Command

Table 27-13, Enter SDR Repository Update Mode Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code

27.20 Exit SDR Repository Update Mode Command

Table 27-14, Exit SDR Repository Update Mode Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code

27.21 Run Initialization Agent Command

This command can be used to cause the Initialization Agent to run. The command can be used to check the status of the Initialization Agent as well.

Table 27-15, Run Initialization Agent

	byte	data field
Request Data	1	[7:1] - reserved [0] - 1b = run initialization agent 0b = get status of initialization agent process
Response Data	1	Completion Code
	2	[7:1] reserved [0] - 1b = initialization completed 0b = initialization in progress

28. FRU Inventory Device Commands

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

The FRU Inventory Device is a 'logical' device. It is not necessarily implemented as a separate physical device, though it can be. For example, the device that contains the SDR Repository Device also typically also holds 'FRU Inventory' information for the main system board and chassis. On the other hand, there may be a separate FRU Inventory device that provides access to the FRU information for a replaceable module such as a Memory Module. Refer to *Appendix G - Command Assignments* for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 28-1, FRU Inventory Device Commands

Command	Section	O/M
Get FRU Inventory Area Info	28.1	M
Read FRU Data	28.2	M
Write FRU Data	28.3	M

O/M = Option/Mandatory for FRU Inventory Devices.

28.1 Get FRU Inventory Area Info Command

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

Table 28-2, Get FRU Inventory Area Info Command

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

28.2 Read FRU Data Command

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

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

Table 28-3, Read FRU Data Command

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

28.3 Write FRU Data Command

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

Table 28-4, Write FRU Data Command

	byte	data field
Request Data	1	FRU Device ID. FFh = reserved.
	2	FRU Inventory Offset to write, LS Byte
	3	FRU Inventory Offset to write, MS Byte
	4:3+N	Data to write
Response Data	1	Completion code 80h = write-protected offset
	2	Count written --- count is '1' based

29. Sensor Device Commands

The following table summarizes the commands that apply to a logical Sensor Device. Refer to *Appendix G - Command Assignments* for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Table 29-1, Sensor Device Commands

Command	Section	O/M
Get Device ID	17.1	M
Cold Reset	17.2	O
Warm Reset	17.3	O[3]
Get Self Test Results	17.4	M[3]
Manufacturing Test Mode On	17.5	O
Broadcast Get Device ID	17.6	M
reserved	-	-
Device Specific Commands	-	-
Get Device SDR Info	29.2	O
Get Device SDR	29.3	O[5]
Reserve Device SDR Repository	29.4	O[5]
Get Sensor Reading Factors	29.5	O[2]
Set Sensor Hysteresis	29.6	O
Get Sensor Hysteresis	29.7	O
Set Sensor Threshold	29.8	O
Get Sensor Threshold	29.9	O[4]
Set Sensor Event Enable	29.10	O
Get Sensor Event Enable	29.11	O[4]
Re-arm Sensor Events	29.12	O[3]
Get Sensor Event Status	29.13	O
reserved	-	-
Get Sensor Reading	29.14	M
Set Sensor Type	29.15	O
Get Sensor Type	29.16	O[4]
Set Event Receiver	23.1	M[1]
Get Event Receiver	23.2	M[1]
Platform Event (a.k.a. Event Message)	23.3	M[1]

Notes:
1. - Mandatory for Event Message Generators only
2. - Mandatory for Non-linear Sensors
3. - Mandatory for manual re-arm Sensors
4.- Mandatory if corresponding 'Set' command is implemented.
5.- Mandatory per information returned in *Get Device SDR Info*

29.1 Static and Dynamic Sensor Devices

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

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

29.2 Get Device SDR Info Command

This command returns general information about the collection of sensors in a Dynamic Sensor Device.

Note: Device Sensor information is LUN based. That is, it is returned individually for each LUN. E.g.. a device could implement four sensors under one LUN, and twelve under another. The SDR Info does *not* return the aggregate of the sensor information. Rather, separate 'Get Device SDR Info' commands need to be issued to each LUN. The 'Device LUNs' field is provided in the response to support this.

Software should assume an allocation unit size of 16-bytes if this command is not implemented.

Table 29-2, Get Device SDR Info Command

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

29.3 Get Device SDR Command

The 'Get Device SDR' command allows the population of sensors for the given LUN in the device to be listed and the associated SDR information for those sensors returned. This is an optional command for Static Sensor Devices, and mandatory for Dynamic Sensor Devices. The format and action of this command is similar to that for the 'Get SDR' command for SDR Repository Devices.

A Sensor Device shall always utilize the same sensor number for a particular sensor. This is mandatory to keep System Event Log information consistent.

Sensor Devices that support the 'Get Device SDR' command return SDR Records that match the SDR Repository formats. See section 37, *Sensor Data Record Formats.*

The 'Get Device SDR' command includes a *Reservation ID* that is used to notify the Requester that a record may have changed during the process of a multi-part read. See *27.11,Reserve SDR Repository,* for more information on the function and use of the Reservation ID field.

Table 29-3, Get Device SDR Command

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

29.4 Reserve Device SDR Repository Command

This command is used to obtain a *Reservation ID*. The Reservation ID is part of a mechanism that is used to notify the Requester that a record may have changed during the process of a multi-part read. See *27.11,Reserve SDR Repository,* for more information on the function and use of Reservation IDs.

Table 29-4, Reserve Device SDR Repository

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Reservation ID, LS Byte 0000h reserved.
	3	Reservation ID, MS Byte

29.5 Get Sensor Reading Factors Command

This command returns the Sensor Reading Factors fields for the specified reading value on the specified sensor. It is used for retrieving the conversion factors for *non-linear* sensors that do not fit one of the generic linearization formulas. See *Non-Linear Sensors* section.

This command is provided for 'analog' sensor devices that are capable of holding a table of factors for linearization, but are incapable of performing the linearization calculations itself. Sensors that produce linear readings, but have non-linear accuracy or resolution over their range can also use this command.

Note: the Response Data is based on the Version and Type of sensor record for the sensor. Only Type 01h record information is presently defined.

Table 29-5, Get Sensor Reading Factors Command

Request Data	1	sensor number (FFh = reserved)
	2	reading byte
Response Data	1	Completion Code
	2	Next reading. This field indicates the next reading for which a different set of sensor reading factors is defined. If the *reading byte* passed in the request does not match exactly to a table entry, the nearest entry will be returned, and this field will hold the *reading byte* value for which an exact table match would have been obtained. Once the 'exact' table byte has been obtained, this field will be returned with a value such that, if the returned value is used as the *reading byte* for the next request, the process can be repeated to cycle through all the Sensor Reading Factors in the device's internal table. This process shall 'wrap around' such a complete list of the table values can be obtained starting with any *reading byte* value.
	3	M: LS 8 bits
	4	[7:6] - M: MS 2 bits [5:0] - Tolerance in +/- ½ raw counts
	5	[7:0] - B: LS 8 bits
	6	[7:6] - B: MS 2 bits Unsigned, 10-bit Basic Sensor Accuracy in 1/100 percent scaled up by unsigned Accuracy exponent. [5:0] - Accuracy: LS 6 bits
	7	[7:4] - Accuracy: MS 4 bits [3:2] - Accuracy exp: 2 bits, unsigned [1:0] - reserved: 2 bits, returned as 00b
	8	[7:4] - R (result) exponent 4 bits, signed [3:0] - B exponent 4 bits, signed

29.6 Set Sensor Hysteresis Command

This command provides a mechanism for setting the hysteresis values associated with the thresholds of a sensor that has threshold based event generation. Hysteresis setting applies to *all* thresholds for the sensor. The positive hysteresis value is used for positive-going thresholds, while the negative going threshold hysteresis value is used for negative-going thresholds. See section *29.13.2, Hysteresis and Event Status* and section *29.13.3, High-going versus Low-going Threshold Events.*

Table 29-6, Set Sensor Hysteresis

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

29.7 Get Sensor Hysteresis Command

This command retrieves the present hysteresis values for the specified sensor. If the sensor hysteresis values are 'fixed', then the hysteresis values can be obtained from the SDR for the sensor.

Table 29-7, Get Sensor Hysteresis

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

29.8 Set Sensor Thresholds Command

This command is used to set the specified threshold for the given sensor. Note that the application issuing this command is responsible for ensuring that thresholds for a sensor are set in the proper order (e.g. that the upper critical threshold is set higher than the upper non-critical threshold).

Table 29-8, Set Sensor Thresholds

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

29.9 Get Sensor Thresholds Command

This command retrieves the threshold for the given sensor.

Table 29-9, Get Sensor Thresholds

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

29.10 Set Sensor Event Enable Command

This command provides the ability to disable or enable Event Message Generation for individual sensor events. The command is also used to enable or disable sensors in their entirety using the *disable scanning* bit.

A typical sensor will come up with Event Messages (EvM) enabled for all thresholds/states. Sensors are not required to have individual or per-event Event Message enables. The type of enable/disable support that a sensor provides can be obtained from the Sensor Data Record for the sensor.

Note that internal event flags and scanning will continue even though Event Message generation is disabled, unless sensor scanning is disabled.

Table 29-10, Set Sensor Event Enable

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

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

* = Devices must accept this command with a variable number (2 to 6) of request data bytes. (In particular, bytes 3 to 6 do not need to be transferred if disabling all Event Messages from the sensor.) This requirement is to allow a reduction in the number of data bytes that must be transferred during the sensor initialization (init agent) process. The receiver shall treat data bytes that are not explicitly transmitted as if they were written as '00h'.

29.11 Get Sensor Event Enable Command

This command returns the enabled/disabled state for Event Message Generation from the selected sensor. The command also returns the enabled/disabled state for scanning on the sensor.

A typical sensor will come up with Event Messages (EvM) enabled for all thresholds. Sensors are not required to have individual or per-event Event Message enables. The type of enable/disable support that a sensor provides can be obtained from the Sensor Data Record for the sensor.

Table 29-11, Get Sensor Event Enable

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

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

* = Devices must accept a variable number of response data bytes (2 to 6). (In particular, bytes 3 to 6 do not need to be transferred if byte 2 indicates that all Event Messages have been disabled.) This requirement is to allow a reduction in the number of data bytes that must be transferred. It is recommended that implementations only return the number of data bytes required to satisfy the command.

29.12 Re-arm Sensor Events Command

This command is provided to re-arm thresholds on sensors that require 'manual' re-arming. It can also be used with sensors that automatically re-arm to cause them to regenerate events when an event condition exists. The re-arm is actually a request for the event status for a sensor to be rechecked and updated.

An initial update in progress bit is provided with the Get Sensor Reading and Get Sensor Event Status commands to help software avoid getting incorrect event status due to a re-arm. For example, suppose a controller only scans for an event condition once every four seconds. Software that accessed the event status using the Get Sensor Reading command could see the wrong status for up to four seconds before the event status would be correctly updated. A controller that has slow updates must implement the initial update in progress bit, and should not generate event messages until the update has completed. Software should ignore the Event Status bits while the initial update in progress bit is set.

Table 29-12, Re-arm Sensor Events

	byte	data field
Request Data	1	sensor number (FFh = reserved)
	2	[7] - 0b = re-arm all event status from this sensor. If 0, following parameter bytes are ignored, but should still be written as 0, if sent. [6:0] - reserved. Write as 000_0000b.
	(3)*	<u>For sensors with threshold based events:</u> [7] - 1b = re-arm assertion event for upper non-critical going high [6] - 1b = re-arm assertion event for upper non-critical going low [5] - 1b = re-arm assertion event for lower non-recoverable going high [4] - 1b = re-arm assertion event for lower non-recoverable going low [3] - 1b = re-arm assertion event for lower critical going high [2] - 1b = re-arm assertion event for lower critical going low [1] - 1b = re-arm assertion event for lower non-critical going high [0] - 1b = re-arm assertion event for lower non-critical going low <u>For sensors with discrete events:</u> [7] - 1b = re-arm assertion event for state bit 7 [6] - 1b = re-arm assertion event for state bit 6 [5] - 1b = re-arm assertion event for state bit 5 [4] - 1b = re-arm assertion event for state bit 4 [3] - 1b = re-arm assertion event for state bit 3 [2] - 1b = re-arm assertion event for state bit 2 [1] - 1b = re-arm assertion event for state bit 1 [0] - 1b = re-arm assertion event for state bit 0
	(4)*	<u>For sensors with threshold based events:</u> [7:4] - reserved. Write as 0000b. [3] - 1b = re-arm assertion event for upper non-recoverable going high [2] - 1b = re-arm assertion event for upper non-recoverable going low [1] - 1b = re-arm assertion event for upper critical going high [0] - 1b = re-arm assertion event for upper critical going low <u>For sensors with discrete events:</u> (00h otherwise) [7] - reserved. Ignore on read. [6] - 1b = re-arm assertion event for state bit 14 [5] - 1b = re-arm assertion event for state bit 13 [4] - 1b = re-arm assertion event for state bit 12 [3] - 1b = re-arm assertion event for state bit 11 [2] - 1b = re-arm assertion event for state bit 10 [1] - 1b = re-arm assertion event for state bit 9 [0] - 1b = re-arm assertion event for state bit 8
	(5)*	<u>For sensors with threshold based events:</u> [7] - 1b = re-arm deassertion event for upper non-critical going high [6] - 1b = re-arm deassertion event for upper non-critical going low [5] - 1b = re-arm deassertion event for lower non-recoverable going high [4] - 1b = re-arm deassertion event for lower non-recoverable going low [3] - 1b = re-arm deassertion event for lower critical going high [2] - 1b = re-arm deassertion event for lower critical going low [1] - 1b = re-arm deassertion event for lower non-critical going high [0] - 1b = re-arm deassertion event for lower non-critical going low <u>For sensors with discrete events:</u> (00h otherwise) [7] - 1b = re-arm deassertion event for state bit 7 [6] - 1b = re-arm deassertion event for state bit 6 [5] - 1b = re-arm deassertion event for state bit 5 [4] - 1b = re-arm deassertion event for state bit 4 [3] - 1b = re-arm deassertion event for state bit 3 [2] - 1b = re-arm deassertion event for state bit 2 [1] - 1b = re-arm deassertion event for state bit 1 [0] - 1b = re-arm deassertion event for state bit 0

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

* = Devices must accept a variable number of request data bytes (2 to 6). This requirement is to allow a reduction in the number of data bytes that must be transferred. The receiver shall treat data bytes that are not explicitly transmitted as if they were written as '00h'.

29.13 Get Sensor Event Status Command

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

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

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

Since the 'Event Status' for a manual re-arm sensor stays until manual cleared, the state is sometime referred to as the 'Event History' or just 'History' for the sensor.

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

29.13.1 Response According to Sensor Type

The response to the *Get Sensor Event Status* command is dependent on the type of event generation for the sensor (threshold based or discrete) and whether the sensor is 'manual re-arm' or 'auto- re-arm'.

If the sensor is 'manual re-arm' then the command returns the latched event status for the sensor. This is essentially those 'flag bits' that indicate that the event had occurred and, if enabled, an event message was generated. A manual re-arm sensor that supports both assertion and deassertion events can have both assertion and deassertion event status set for a state simultaneously.

If the sensor is 'auto- re-arm' then the command returns unlatched present event status for the sensor. The event status for auto- re-arm sensors can be derived from the present status information returned in a Get Sensor

Reading command, if the hysteresis values are known. For this reason, the Get Sensor Event Status command is typically not implemented for auto- re-arm sensors. Instead, if system management software needs to determine event status, it derives it from the *Get Sensor Reading* and hysteresis settings.

The format of the *Get Sensor Event Status* response is dependent on whether the sensor was threshold based or discrete.

Table 29-13, Get Sensor Event Status Response Overview

Sensor Class	Auto- re-arm	Status Returned
Threshold based	Yes	Present threshold comparison event status. This is redundant to the threshold comparison status returned with the 'Get Sensor Reading' command if the sensor has no hysteresis. Otherwise, software can derive the event status from the Get Sensor Reading command if it knows the hysteresis value.
	No	Latched threshold comparison status. Since manual re-arm status is 'sticky', the status may be different than the comparison status returned with the 'Get Sensor Reading' command.
Discrete	Yes	Present event status represented by a bit mask indicating the event conditions that are presently active on the sensor. Note: this is redundant to the status returned with the 'Get Sensor Reading' command if there is no hysteresis associated with the sensor.
	No	Latched event status represented by a bit mask indicating the event conditions that have been detected on the sensor. Since manual re-arm status is 'sticky', the status may be different than the comparison status returned with the 'Get Sensor Reading' command.

29.13.2 Hysteresis and Event Status

For threshold-based sensors the event status reflects whether the sensor is armed (ready to generate another event) or not. This means that there is a difference between the event status, returned by this command, and the *comparison status* returned by the *Get Sensor Reading* command. For example, suppose a sensor has an upper non-recoverable threshold with a threshold value of 98h and a positive-going threshold hysteresis value of 2. That sensor's event status (byte 3, bit 5, below) would get set when the reading hit 98h, but would not clear until the reading hit 95h. (a 0 hysteresis would yield a re-arm point of 97h, therefore a positive-going hysteresis of 2 corresponds to a re-arm point of 95h).

A sensor can only return a '1' for the assertion or deassertion events that it supports. If a sensor does not support particular assertion or deassertion event states it must always return a '0' for the bits associated with those states. For example, suppose a sensor supports assertion events for discrete state 0, but does not support deassertion events. The sensor will set the state 0 assertion event status to 1 when the event becomes asserted and to 0 when the event condition clears, but the state 0 deassertion event status bit will always be 0. This operation is specified so that a sensor does not return an 'event occurred' status for states that can not generate an Event Message.

29.13.3 High-going versus Low-going Threshold Events

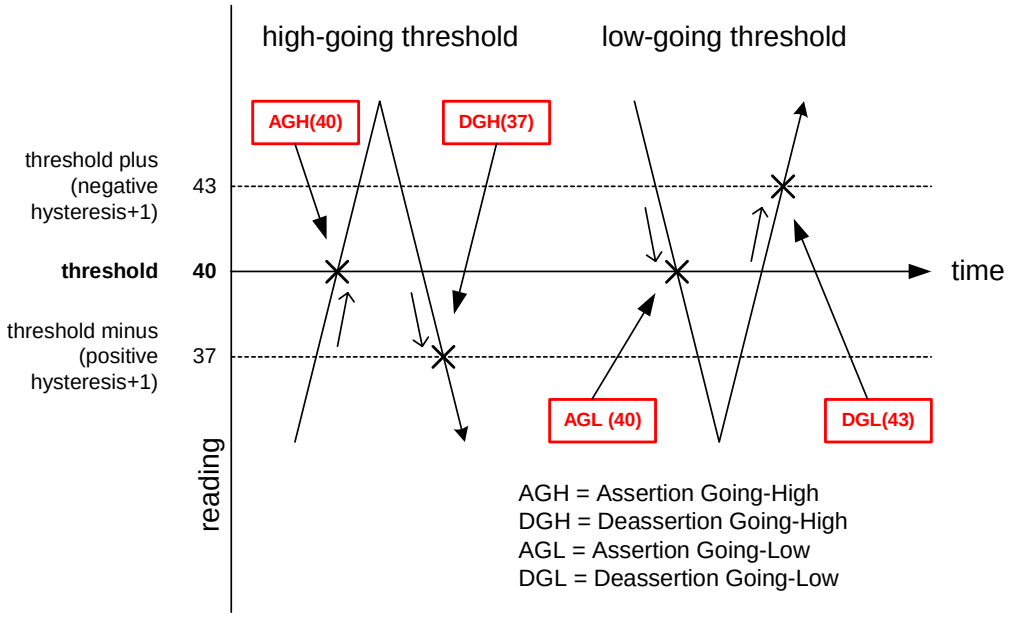
The differences between high-going and low-going threshold events are in what direction the reading needs to be going for an event to occur, in where deassertion events occur, and in how hysteresis affects where deassertion events occur. *Figure 29-1, High-Going and Low-Going Event Assertion/Deassertion Points*, illustrates these differences.

A high-going threshold has its assertion events become set when the reading is ≥ the threshold, while for a low-going event the assertion event becomes set when the reading is ≤ the threshold. Even more difference is seen with where the de-assertion events occur. A high-going threshold must have the reading drop to a value that is positive_hysteresis+1 counts *below* the threshold value in order for the deassertion event to occur (and for the assertion event status to clear). A low-going threshold must have the reading rise to negative_hysteresis+1 counts *above* the threshold to become deasserted.

Note that a zero hysteresis value still leads to a difference between where the deassertion events occur. An event can't be in the asserted and deasserted states simultaneously. Thus, for zero hysteresis a high-going threshold event becomes asserted when the reading is ≥ the threshold, and becomes deasserted when the reading goes ≤ the threshold *minus* one. A low-going threshold event becomes asserted when the reading goes ≤ the threshold, and becomes deasserted when the reading goes ≥ the threshold *plus* one.

A system implementation will typically only use either high-going or low-going events for a given threshold, but not both simultaneously.

Figure 29-1, High-Going and Low-Going Event Assertion/Deassertion Points



29.13.4 Get Sensor Event Status Command Format

The following table shows the format of the Get Sensor Event Status command.

Table 29-14, Get Sensor Event Status Command

Request Data	1	Sensor number (FFh = reserved)
Response Data	1	Completion Code
	2	[7] - 0b = All Event Messages disabled from this sensor [6] - 0b = Sensor scanning disabled [5] - 1b = initial update in progress. This bit is set to indicate that a 're-arm' or 'Set Event Receiver' command has been used to request an update of the sensor status, and that update has not occurred yet. Software should use this bit to avoid getting an incorrect status while the first sensor update is in progress. This bit is only required if it is possible for the controller to receive and process a 'Get Sensor Reading' or 'Get Sensor Event Status' command for the sensor before the update has completed. This is most likely to be the case for sensors, such as fan RPM sensors, that may require seconds to accumulate the first reading after a re-arm. [4:0] - reserved. Ignore on read.

3	<u>For sensors with threshold based events:</u> (High-going events are asserted when value first becomes ≥ threshold. Low-going events are asserted when value first becomes ≤ corresponding threshold.) [7] - 1b = assertion event condition for upper non-critical going high occurred [6] - 1b = assertion event condition for upper non-critical going low occurred [5] - 1b = assertion event condition for lower non-recoverable going high occurred [4] - 1b = assertion event condition for lower non-recoverable going low occurred [3] - 1b = assertion event condition for lower critical going high occurred [2] - 1b = assertion event condition for lower critical going low occurred [1] - 1b = assertion event condition for lower non-critical going high occurred [0] - 1b = assertion event condition for lower non-critical going low occurred <u>For sensors with discrete events:</u> [7] - 1b = state 7 assertion event occurred [6] - 1b = state 6 assertion event occurred [5] - 1b = state 5 assertion event occurred [4] - 1b = state 4 assertion event occurred [3] - 1b = state 3 assertion event occurred [2] - 1b = state 2 assertion event occurred [1] - 1b = state 1 assertion event occurred [0] - 1b = state 0 assertion event occurred
(4)*	<u>For sensors with threshold based events:</u> [7:4] - reserved. Write as 0000b. [3] - 1b = assertion event condition for upper non-recoverable going high occurred [2] - 1b = assertion event condition for upper non-recoverable going low occurred [1] - 1b = assertion event condition for upper critical going high occurred [0] - 1b = assertion event condition for upper critical going low occurred <u>For sensors with discrete events:</u> (00h otherwise) [7] - reserved. Ignore on read. [6] - 1b = state 14 assertion event occurred [5] - 1b = state 13 assertion event occurred [4] - 1b = state 12 assertion event occurred [3] - 1b = state 11 assertion event occurred [2] - 1b = state 10 assertion event occurred [1] - 1b = state 9 assertion event occurred [0] - 1b = state 8 assertion event occurred

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

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

29.14 Get Sensor Reading Command

This command returns the present reading for sensor. The sensor device may return a stored version of a periodically updated reading, or the sensor device may scan to obtain the reading after receiving the request.

The meaning of the state bits returned by Discrete sensors is based on the Event/Reading Type code from the SDR for the sensor. This can also be obtained directly from the controller if the optional *Get Sensor Type* command is supported for the sensor.

Table 29-15, Get Sensor Reading Command

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

29.15 Set Sensor Type Command

This command is used to assign the Sensor Type and Event/Reading Type to a specified sensor. A management controller that implements sensors and generates events for those sensors must return the Sensor Type and Event/Reading Type in the Event Messages from those sensors. By allowing those values to be assigned, it is possible to create a 'generic' management controller with fixed firmware that is field configured with sensor type, reading, and event type information for the particular application.

For example, a controller could provide a set of unassigned 'digital' discrete sensors. The *Set Sensor Type* command could allow one of these sensors to be set as a 'Processor' sensor that returns 'Inserted/Removed' status, and generates an event on 'Removed'. Another sensor could be assigned to be a Physical Security (Chassis Intrusion) sensor the returns

The same approach could be used to assign monitoring functions to a controller that provides A/D inputs and an associated set of unassigned threshold-based analog sensors. One sensor could be assigned to be a voltage sensor, while another could be assigned to be a temperature sensor, etc.

The Sensor Data Records include an 'Init Sensor Type' bit that indicates whether this information should be assigned to the controller as part of the Initialization Agent process.

The controller can implement this command such that the assignment is volatile or non-volatile. A non-volatile assignment would allow the assignment to be retained across power cycles or system resets, at the cost of providing non-volatile storage for the controller. A controller that has a volatile assignment would rely on the Initialization Agent function to assign the sensor type. This trade-off with this approach is that there would be more times when the sensor was disabled and unassigned prior to the execution of the Initialization Agent process.

Table 29-16, Set Sensor Type Command

Request Data	1	sensor number (FFh = reserved)
	2	sensor type (per *Table 36-3, Sensor Type Codes*)
	3	[7] - reserved [6:0] - Event/Reading type code (per *Table 36-2, Generic Event/Reading Type Codes*)
Response Data	1	Completion Code.

29.16 Get Sensor Type Command

This command is used to retrieve the Sensor Type and Event/Reading Type for the specified sensor. This command is mandatory for sensors that respond to the *Set Sensor Type* command.

Table 29-17, Get Sensor Type

Request Data	1	sensor number (FFh = reserved)
Response Data	1	Completion Code.
	2	sensor type (per *Table 36-3, Sensor Type Codes*)
	3	[7] - reserved. [6:0] - Event/Reading type code (per *Table 36-2, Generic Event/Reading Type Codes*)

30. Sensor Types and Data Conversion

Sensors can be generally classified into two types, Linear/Linearizable Sensors and Non-Linear Sensors. The difference between the two types is mainly in the manner in which software that accesses the sensors needs to handle the conversion of the sensor readings.

Sensor Devices are allowed to implement their sensors using 'raw' values for their thresholds and for returning their readings. For example, the physical device that implements a voltage sensor will often be an A/D converter. The values that the Sensor Device returns will typically be in A/D counts, rather than an direct integer value in volts.

Therefore, software that interfaces to these values must know how to convert and interpret these values. The 'conversion factors' for these values shall either be provided in the Sensor Data Record for the sensor, or shall be retrievable from the Sensor Device. In the example of the previous paragraph, the Sensor Data Record for the voltage sensor would contain values that allow software to convert those A/D counts to a voltage value.

Allowing the sensor values to be isolated from the measurement units allows the Sensor Devices themselves to be implemented in a simpler manner. The measured quantity can also be changed or measurement adjusted by changing the Sensor Data Record without having to change the physical implementation of the Sensor Device. A physical instantiation of a Sensor Device that implement a sensor that's an A/D converter could have that sensor defined as a voltage measurement sensor in one system implementation, and a current sensor in another.

30.1 Linear and Linearized Sensors

Linear sensors return readings that can be converted to the desired sensor units (temperature, voltage, etc.) using a linear conversion formula. Linearized sensors are sensors that can have one of a set of pre-specified conversion formulas applied to the reading to linearize it. After linearization the sensor reading can be converted to final units as if it was linear in the first place.

Linear and Linearized sensors are also considered as having *constant* Accuracy, Tolerance, and Resolution over the range of the *raw readings* from the sensor. Thus, for a linearized sensor, the effects of accuracy, tolerance, and resolution are to be applied *prior* to application of the linearization formula.

30.2 Non-Linear Sensors

Non-linear sensors are sensors that either cannot be linearized using one of the pre-determined linearization formula, or are sensors that do not have constant conversion factors, accuracy, tolerance, and/or resolution over the range of their raw readings.

Because the conversion factors, accuracy, etc., can vary - System Management Software must treat non-linear sensors by obtaining these factors for the reading of interest by querying the sensor using a 'Get Sensor Reading Factors' command. This means that polling of non-linear sensors is a two-step process. First, System Management Software obtains the raw reading from the sensor, second it issues a 'Get Sensor Reading Factors' command to retrieve the conversion factors for that reading.

Since the conversion factors, accuracy, tolerance, etc., are returned with the reading, a linearization function is not applied in the conversion.

30.3 Sensor Reading Conversion Formula

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

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

where:

x Raw reading

y Converted reading

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

M Signed integer constant multiplier

B Signed additive 'offset'

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

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

Accuracy, tolerance, M, and B for 'Non-linear' sensors are only valid at the nominal reading. Otherwise, these factors must be obtained by 'querying' the sensor for these factors at the reading of interest using the 'Get Sensor Reading Factors' command. Refer to *Section 29.5, Get Sensor Reading Factors,* for more information.

30.4 Resolution, Tolerance and Accuracy

Resolution, Tolerance, and Accuracy are applied to the RAW reading for Linear and Linearizable sensors, prior to the application of any further conversion formula.

30.4.1 Tolerance

Tolerance is specified in the Sensor Data Records in +/- ½ raw counts. The +/- implies that the tolerance value is '0' based. There is no 'B' offset used in converting the tolerance value to units. Tolerance can thus be converted to the to units using the formula $y = L[Mx/2 * 10^{K2}] \text{ units}$. Where L, M, and K2 are as specified above.

Note that tolerance can vary at each reading for a non-linear sensor. The 'Get Sensor Reading Factors' command can be used to obtain the tolerance at a given reading.

30.4.2 Resolution

Resolution indicates the separation in units between successive raw reading values. For linear sensors, resolution is obtained from the 'M' factor. To convert M to resolution in units, use the formula $y = abs(M * 10^{K2}) \text{ units}$. Where abs( ) means use the absolute value.

30.4.2.1 Resolution for Non-linear & Linearizable Sensors

The resolution typically varies at each different reading value of a non-linear or linearizable sensor. One approach to determining a resolution for these types of sensors is to examine the points neighboring the reading of interest. I.e. for reading 'x', take the difference between x and x+1 converted to units as the resolution in the 'positive' direction, and the difference between x and x-1 converted to units as the resolution in the 'negative' direction.

30.4.2.2 Offset Constant Relationship to Resolution

There is a relationship between the constant offset factor, 'B', and resolution. For example, if a voltage sensor has a resolution of 100 mV and a zero 'B' offset, the raw readings and threshold settings from the sensor would convert to units in even increments of 100 mV. E.g. starting from 0: 0.000V, 0.100V, 0.200V, etc. If the same sensor had a 'B' offset equivalent to 2.250V, then the same readings would be 2.250V, 2.350V, 2.450V, etc.

This may need to be incorporated into the user interface for a management application that displays readings, or allows thresholds to be set. For example, the user interface could provide a scrolling selection for threshold settings in units that incremented and decremented according to the specified offset and resolution values.

30.5 Management Software, SDRs, and Sensor Display

Analog sensor devices are always accessed using raw values, where the term 'raw values' refers to the fact that values from the sensor are not in the final assigned units. For example, an analog-to-digital converter returns a number that is representative of a voltage applied to the device. Whether that voltage represents temperature or fan speed is dependent on how the device is applied.

If a *Get Sensor Reading* command is issued to a management controller that provides the A/D converter reading, the management controller will typically just return the direct reading from the converter. Software uses the conversion factors in the SDR for the sensor to convert that reading to units that the platform vendor or system integrator selected as being appropriate for the device. The selection of one type of unit versus another is typically made according to what best fits the monitoring hardware and maintains the best accuracy.

Thresholds and threshold comparisons are also done with raw values. The threshold designations "Upper" and "Lower" are solely with respect to the hardware that is doing the comparison of the thresholds with the raw reading values. Correspondingly, the Upper and Lower threshold values in the SDRs reflect these raw values. The upper threshold is always for the raw reading with the most positive value, the lower threshold with the most negative value. This is not necessarily the value with the greatest absolute magnitude. E.g. if I have two threshold values, 5 and 10, 5 would be the lower threshold and 10 would be the upper. If I have -5 and 10, -5 would be the lower, and 10 would be the upper. If I have -5 and -10, however, -10 would be the lower threshold and -5 the upper - since -5 is more positive than -10.

30.5.1 Software Display of Threshold Settings

In most cases, software can directly display thresholds directly after performing the conversion to units. There is a set of cases, however, where it is recommended that software display the thresholds with the upper and lower threshold names 'swapped' in order to provide a more intuitive user interface. This can occur with sensors that have a 1/x Linearization factor, as described in the following example.

Suppose I have a 'tach fan' that outputs a number of pulses per fan revolution, and a sensor that returns a raw reading that is directly proportional to the period (interval) between the pulses. As the fan speeded up, the reading would decrease (since the interval between pulse would get smaller) and as the fan slowed down the pulse period would increase. The following shows some example 'raw value' thresholds that might be returned by such a sensor.

Sensor Hardware Settings (raw units):

Upper Critical going high Threshold = 100 (Fan going too slow)

Lower Critical going low Threshold = 10 (Fan going too fast)

An SDR could be made that directed software to convert and report this period in terms of seconds. For example, the SDR could contain linear conversion factors that converted the reading into seconds. In this example, assume a reading of 100 corresponds to 100 milliseconds, and a value of 10 corresponds to 10 milliseconds. The user interface may then present the following information:

Software User Interface Display:

Sensor Type: Fan

Upper Critical Threshold: 100 ms

Lower Critical Threshold: 10 ms

While correct, this display is not particularly intuitive to the end user. The user would probably make more sense of the values if they were given in RPM. Thus, a more 'user friendly' SDR would direct software to present the reading as RPM. To do this requires converting the raw reading using a 1/x linearization factor, either in the SDR or done by the system management software. In this example, assume the fan puts out 1 pulse per revolution. A 10 ms interval would correspond to 1 revolution in 10 ms, which equals 100 revolutions per second, or 6000 revolutions per minute (6000 RPM). Similarly, a 100 ms interval would correspond to 600 RPM.

In this case, a piece of user interface software that displayed the threshold settings directly by applying the conversion factors would see:

Software User Interface Display:

Sensor Type: **Fan**

Upper Critical Threshold: **600 RPM**

Lower Critical Threshold: **6000 RPM**

While this is correct with respect to the management controller that is doing the comparisons, *an end user may be confused to see a Lower Critical Threshold value that is greater than the Upper Critical Threshold.* Thus, it's recommended that the *User Interface* swap the threshold names when the 1/x factor is encountered. This will allow the end user to see the thresholds presented as:

Software User Interface Display:

Sensor Type: **Fan**

Upper Critical Threshold: **6000 RPM**

Lower Critical Threshold: **600 RPM**

This presentation of fan speed thresholds is likely to make more sense to the typical user.

30.5.2 Notes on Displaying Sensor Readings & Thresholds

The following should be kept in mind when designing software that utilizes SDRs for the display for sensor readings and thresholds:

- 'Analog' sensor readings and thresholds use raw values.

- Thresholds in sensors and SDRs are given in raw values.
- The management controller performs its threshold comparisons are done against raw values. Changing the SDR has no effect on the meaning of Upper and Lower as far as the management controller is concerned.
- Changing the units and conversion factors for a sensor does not change the hardware behavior.
- Sensor Data Records tell software how to convert the raw values into units that the platform vendor or system integrator deemed appropriate for the sensor. The values are typically selected based on what provides the most accurate or direct conversion of the hardware.
- System software can elect to convert units for display to the user. For example, system software may elect to display all temperatures in Fahrenheit, even if the Sensor Data Record provides factors for converting the reading to Celsius.
- To make the display more intuitive to the user, it's recommended that software swap the threshold names when linearization or conversions factors, e.g. 1/x, cause the sense of 'Upper' and 'Lower' to be reversed during the conversion from raw values to display units.
- The System Event Log also returns values in raw units, and thresholds that are related to raw units. System Events are best displayed when the SEL Record information is combined with the SDR information for the sensor. The meaning of Upper and Lower threshold events cannot be fully understood without using the SDR information. For example, it's possible that an 'Upper Critical' temperature event could actually correspond to a LOW TEMPERATURE. Thus, if a System Event Log display utility doesn't have access to the SDR information, it's best to emphasize the *criticality* and sensor type associate with the event rather than what particular threshold was crossed.

31. Timestamp Format

Timestamping is a key part of event logging and tracking changes to the Sensor Data Records and the SDR Repository. The following specifies the format of the seconds-based timestamp used in this document.

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

The timestamps used for SDR and SEL records are assumed to be specified in relative local time. That is, the difference between the timestamp does not include the GMT offset. To convert the timestamp to a GMT-based time requires adding the GMT offset for the system. (The GMT offset needs to be obtained from system software level interfaces, there is no provision in the IPMI commands for storing or returning a GMT offset for the system.) Applications may use ANSI C time standard library routines for converting the SEL timestamp reading into other time formats. Be aware that this may require additional steps to account for the system's GMT offset.

31.1 Special Timestamp values

0xFFFFFFFF indicates an invalid or unspecified time value.

0x00000000 through 0x20000000 are used for timestamping events that occur after the initialization of the System Event Log device up to the time that the timestamp is set with the system time value. Thus, these timestamp values are relative to the completion of the SEL device's initialization, *not* January 1, 1970.

32. Accessing FRU Devices

FRU devices can either be located behind a management controller or located directly on the IPMB. The sensor data records include a *FRU Device Locator* record that tells software where the device is located and what type of commands are required to access the FRU device. FRU devices can located in three different types of location:

- **Behind a management controller** and accessed using *Read/Write FRU Device* commands. Multiple FRU devices can be behind a management controller. The *Read/Write FRU Devic*e commands include a *FRU Device ID* field that is used to identify individual FRU devices on the given LUN in the management controller. Up to 255 FRU devices can be located on a given LUN. FRU Device ID #00 at LUN 00b is pre-defined as being the FRU Device for the FRU that the management controller is located on. Since there are four possible LUNs for a management controller, this means up to 255*4 FRU devices can be supported behind a single management controller using this mechanism. The *Read/Write FRU Device* commands provide an abstracted interface that hides the technology used to implement the FRU device from system software.
- **SEEPROM on a private bus** behind a management controller. These devices are accessed using *Master Write-Read* commands. System software needs to know the operation of a 24C02-compatible SEEPROM interface to access these devices.
- **SEEPROM on the IPMB**. These devices are typically accessed using a *Master Write-Read* command to the IPMB via the BMC. System software needs to know the operation of a 24C02-compatible SEEPROM interface to access these devices. Note that there are only eight IPMB addresses available for typical 24C02-type SEEPROM devices, four of which are reserved for the baseboard supplier. (Refer to the *IPMB Address Allocation* specification).

The FRU Device Locator record provides fields that identifies the type of location for the FRU Device and where it’s located. The following table illustrates how these fields are used.

Table 32-1, FRU Device Locator Field Usage

FRU Device Type and Location	FRU Device Locator Fields		Access Method
FRU Device Accessed via Read/Write FRU commands to management controller	Device Access Address:	IPMB Slave address of the controller that accepts the *Read/Write FRU Device* commands for access the FRU Device.	*Read / Write FRU Device* commands to management controller providing access to the FRU Device.
	FRU Device ID / Device Slave Address:	FRU Device ID.	
	Access LUN / Bus ID:	bit 7 = 1 indicates device is access using *Read/Write FRU Device* commands. bits 4:3 hold the LUN to send the *Read/Write FRU Device* commands to. bits 2:0 = 000b. (no private bus ID)	
SEEPROM On private bus behind a management controller	Device Access Address:	IPMB slave address of the controller to send the *Master Write-Read* command to.	*Master Write-Read* command to management controller that provides access to the private bus
	FRU Device ID / Device Slave Address:	Slave address of the SEEPROM on the private bus. Used in the *Master Write-Read* command.	
	Access LUN / Bus ID:	bit 7 = 0 indicates device is a non-intelligent device. bits 4:3 hold the LUN to send the *Master Write-Read* I^2C command to. bits 2:0 hold the Private Bus ID to use in the *Master Write-Read* command.	
SEEPROM Device directly on IPMB	Device Access Address:	00h. Indicating device is directly on IPMB	*Master Write-Read* command through BMC from system software, or access via other interface providing low-level I^2C access to the IPMB.
	FRU Device ID / Device Slave Address:	Slave address of the SEEPROM on the IPMB.	
	Access LUN / Bus ID:	bit 7 = 0 indicates device is a non-intelligent device. bits 3:0 = 0h indicates device is on the IPMB.	

33. Using Entity IDs

An *Entity ID* is a standardized numeric code that is used in SDRs to identify the types of physical entities or FRUs in the system. The codes include values for entities such as Processor, Power Supply, Fan, etc. The Entity ID values are specified in *Table 37-12, Entity ID Codes*.

The Entity ID is associated with an *Entity Instance* value that is used to indicate the particular instance of an entity. For example, a system with four processors would use an Entity Instance value of '0' to identify the first processor, '1' for the second, and so on. [Note: The assignment of Entity Instance values is up to the implementer. There's no predefined semantics on the Entity Instance other than its use to differentiate among Entities of the same type. For example, an implementer could designate the first processor using Entity Instance 0, 1, or 12...]

The SDR for a sensor includes Entity ID and Entity Instance fields that identify the entity associated with the sensor. This allows system software to tell a temperature sensor associated with 'Processor 1' from a temperature sensor associated with 'Power Supply 2'. The use of numeric codes facilitates the development of automated applications that act on this relation information. It also supports internationalization and localization.

33.1 System- and Device-relative Entity Instance Values

Entity Instance values can be in one of two ranges, system-relative or device-relative. In IPMI v1.0, all Entity Instance values were system-relative - meaning that the Entity Instance numbers for a given had to be unique for all entities in the system sharing the same Entity ID. A problem with this approach is that add-in cards and management controllers cannot have pre-assigned Entity Instances because of the potential that those values would overlap with Entity Instance values already present in the system.

In order to correct this situation, the IPMI v1.5 specification splits the Entity Instance value into two ranges. Entity Instance values in the system-relative range are required to be unique for all entities with the same Entity ID in the system. Device-relative Entity Instance values are only required to be unique among all entities that have the same Entity ID within a given device (management controller). For example, management controller 'A' and 'B' could both have FAN entities that have and Entity Instance value of '60h'.

The system-relative Entity Instance definition matches the original Entity Instance definition in IPMI v1.0. Therefore an IPMI v1.0 implementation that is being migrated to IPMI v1.5 does not need to change Entity Instance values if they're already in the system-relative range.

Table 33-1, System and Device-Relative Entity Instance Values

Range	Name	Definition
00h-5Fh	system-relative	The Entity Instance number must be unique for each different entity of the same type Entity ID in the system.
60h-7Fh	device-relative	7Fh = Device-relative. Instance number is unique for each different entity of type Entity ID for the management controller device that provides access to the sensors for the entity. The entity is uniquely identified by the combination of the Sensor Device number and the Entity Instance number. It is recommended that console software subtract 60h when presenting device-relative Entity Instance values, and present the Entity Instance number along with an ID for the device providing the interface to the entity. For example, suppose management controller '1' had a FAN entity with a device-relative Entity Instance value of 61h. It may make more sense to the user to refer to the entity as "Controller 1, Fan 1" than 'Controller 1, Fan 61h'. Entities with system-relative Entity Instance values could be preceded with the word 'System' (or something similar). E.g. "System, Fan 1."

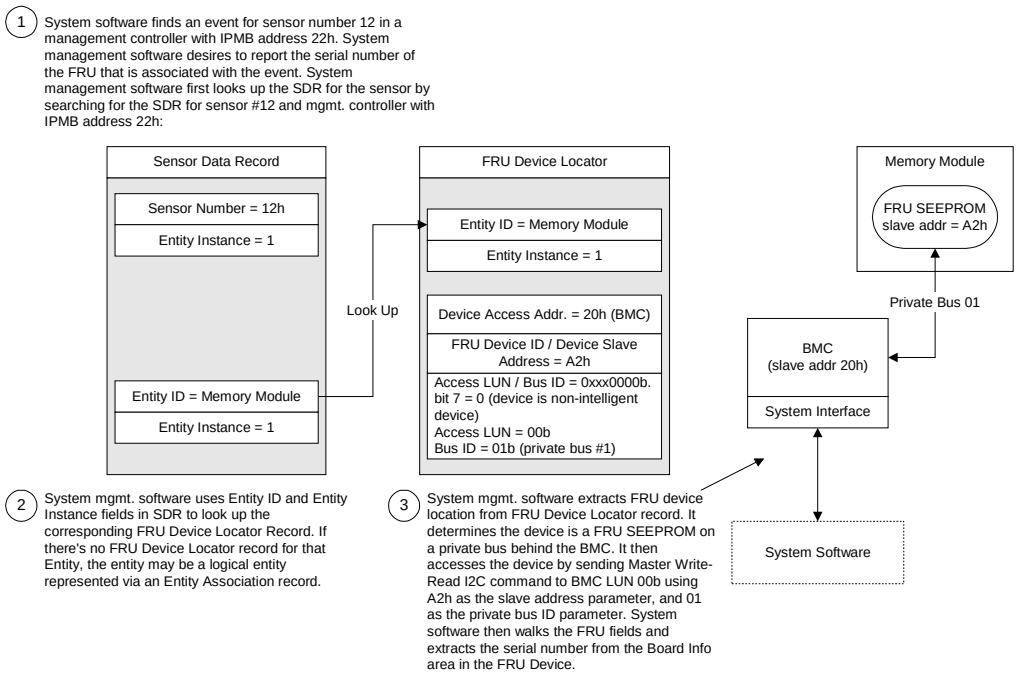
33.2 Restrictions on Using Device-relative Entity Instance Values

Note that when using a Sensor Device Relative instance number, all sensors monitoring a given entity must be provided via one management controller. Otherwise, it may appear that there are more instances of the entity than are actually present. For example, suppose an add-in card has a single fan where current is monitored via sensor 1 in management controller 'A' and the speed monitored via sensor 2 in management controller 'B'. Because the management controller's ID is now part of what uniquely identifies the entity, software would view these sensors as monitoring different fans, even if the Sensor Device Relative instance number were the same.

33.3 Sensor-to-FRU Association

A given Entity may itself be a FRU. In this case the Entity ID can be used to look up the FRU Device Locator record for that Entity. (The FRU Device Locator record is a type of SDR that identifies the location and type of FRU Devices in the platform management subsystem). The following figure presents an example illustrating the steps that system management software could use to locate and access the FRU for the Entity associated with a sensor.

Figure 33-1, Sensor to FRU Lookup



34. Handling Sensor Associations

This section presents information on handling the relationships between Entities and sensors.

34.1 Entity Presence

Multiple sensors can be associated with the same Entity. For example, a voltage sensor, temperature sensor, and processor sensor can all be associated with the same Processor 1 entity. This forms a simple association among the sensors.

Some entities, such as processors, have sensor-specific discrete sensor types that include presence as part of their definition. If the Entity has a presence status associated with it, system management software should ignore sensors and other status associated with the Entity when the Entity is absent. *A bit in the SDR for each sensor indicates whether the sensor should be ignored when the associated Entity is absent or disabled.*

For other entities, an *Entity Presence* sensor can be defined. This sensor is always implemented as a generic discrete sensor using an Event/Reading Type code of 08h, for Device Absent/Device Present, or 09h for Device Enabled/Disabled.

34.2 Software detection of Entities

This section describes steps that software can use to detect the presence of an Entity. There are several ways to detect the presence of an Entity. Software can determine that an Entity is present by the existence of a FRU device for the entity, active sensors for that entity, by the state of a 'presence' bit for the sensor, or by an Entity Presence sensor for the entity:

- An Entity is present or absent if there is an active sensor that holds an explicit present bit or presence sensor that indicates the presence/absence of the Entity. Presence/absence bits and presence sensors take precedence over all other presence determination mechanisms.
- An Entity is present if the FRU Device for that Entity exists (can be accessed) - unless overridden by an explicit presence bit or Entity Presence sensor indicating that the entity is absent.
- An Entity should be assumed absent if the Entity Presence sensor for the entity is inaccessible, or the Entity is part of an Entity Association under an inaccessible Entity Presence sensor. This provision allows the entity presence sensor to represent the presence of a collection of entities. Software should then ignore any sensors associated with absent entities.
- An Entity is present if there is at least one active sensor for the Entity (and there is no explicit sensor saying the Entity is 'absent'). A sensor is 'active' if scanning is enabled. For each of these sensors, check to see that at least one of the sensors is scanning by checking the "sensor scanning disabled" bit via the Get Sensor Reading command. Per section 11.5, software should ignore this bit if its set to 'disabled'. If there are no active sensors for the entity, then it should be assumed that the Entity is absent.
- An Entity is assumed present if the entity is a 'container' entity in an entity association, and at least one of the 'contained' entities is present. For example, suppose there is a 'power unit' entity that is a container entity for three power supply entities. Then the power unit entity is present if any of the power supply entities are present.

Thus, the steps to detecting an Entity are:

a) Scan the SDRs for sensors associated with the entity.

b) If there is an active sensor that includes a presence bit, or the entity has an active Entity Presence sensor, use the sensor to determine the presence of the entity.

c) Otherwise, check to see that there is at least one active sensor associated with the entity. Do this by doing 'Get Sensor Readings' to the sensors associated with the entity until a scanning sensor is found.

d) If there are no active sensors directly associated with the entity, check the SDRs to see if the entity is a container entity in an entity-association. If so, check to see if any of the contained entities are present, if so, assume the container entity exists. Note that this may need to be iterative, since it's possible to have multi-level entity associations.

e) If there are no active sensors for the entity, and the entity is not the container entity in an active entity-association, then the entity is present is there is a FRU device for the entity, and the FRU device is present.

It should not be considered an error if a FRU device locator record is present for a FRU device, but the FRU device is not there. This is because there may be a presence sensor that indicates the FRU is absent. It would only be an error if there was an active sensor or entity-association that was associated with the Entity, and the FRU device for the Entity wasn't there.

34.3 Using Entity Association Records

Entity Association Records allow multiple physical entities to be grouped into a single logical entity. (Note, the structure also allows logical entities to be comprised of other logical entities...) This grouping is typically used for defining a logical entity that has sensing associated with it that relates to the physical entities that comprise the group.

For example, in a system with redundant power supplies, a Power Unit can be viewed as a logical entity comprised of multiple Power Supply entities. The Entity Association Record can be used to tell system management software which Power Supply entities make up the power unit. The association also allows system management software to correlate failures from individual entities with status of the logical entity.

Continuing with the Power Unit example, suppose the Power Unit had a discrete sensor that returned Redundancy Status (Redundant, Redundancy Degraded, Non-Redundant, etc.), and that each individual Power Supply had a sensor that returned a failure status. System management software could use the association to correlate a change in the Power Unit redundancy status with a corresponding failure of a Power Supply in the Power Unit.

The Entity Association record also allows physical entities to be grouped to indicate that a single sensor applies to multiple entities. For example, a voltage regulator could be shared among pairs of processors. Logical grouping of entities can be handled in one of two ways. The first is to use bit 7 in the Entity Instance field. This bit indicates whether a particular entity should be viewed as a logical ‘group’ entity rather than by its base definition in the Entity ID table.

For example, if the Entity was ‘Processor 1’, setting this bit would tell software to consider the entity as ‘Processor Group 1’ instead. The Entity Association record corresponding to this logical entity could then consist of contained processor entities (by convention, a logical processor group only contains processor entities - though this is not mandatory.)

If this way of identifying a grouping is not sufficient, there is a generic ‘Group’ Entity ID that is provided solely to serve as the container entity for grouping of entities that do not fall under a specific logical Entity ID type.

It is thus possible to present several types of relationships between sensors and entities. With a direct Sensor-to-entity relationship (no Entity Association record involved) a typical sensor-to-entity relationship would be:

“Power Supply 1 Temperature Sensor ”

While with an Entity Association for a pre-defined logical entity a possible relationship would be:

"Redundancy Sensor for Power Unit 1 containing Power Supply 1, Power Supply 2, and Power Supply 3"

The Entity Association for a logical group relationship would be:

"Voltage Sensor for Processor Group 1 containing Processor 1 and Processor 2"

Lastly, the Entity Association, a generic Group relationship could be:

"Voltage Sensor for Group 2 containing Processor 1 and Fan 2"

It is recommended that console software subtract 60h when presenting device-relative Entity Instance values, and present the Entity Instance number along with an ID for the device providing the interface to the entity. For example, suppose management controller '1' had a FAN entity with a device-relative Entity Instance value of 61h. It may make more sense to the user to refer to the entity as "Controller 1, Fan 1" than 'Controller 1, Fan 61h'. Entities with system-relative Entity Instance values could be preceded with the word 'System' (or something similar). E.g. "System, Fan 1."

35. Sensor & Event Message Codes

This section provides a description of the Sensor and Event Codes and the manner in which they are used within Sensor Data Records and Event Messages. Following is a series of tables that define these codes.

35.1 Sensor Type Code

Each sensor has a *Sensor Type Code*. These codes are defined in *Table 36-3, Sensor Type Codes*. Sensor Type Codes are used both in SDRs and Event Messages. An example of a Sensor Type Code is code 01h, which indicates a *Processor* sensor.

35.2 Event/Reading Type Code

The Event/Reading Type Code is used for specifying the types of readings that a sensor can provide, the types of events that a sensor can generate, and the type of state or transition that triggered an event.

Event/Reading Type Codes are split into three categories:

Generic

The Event/Reading Type Code specifies one of a set of pre-defined enumerations that are applicable to many different types of sensors. It is expected that system software will have a-priori knowledge of how to interpret these types of states and events. As such, these enumerations should be used whenever possible. For example, an Event/Reading Type Code of 02h identifies the “DMI Usage State” enumeration, which consists of three members “(transition to) Idle”, “(transition to) Active”, and “(transition to) Busy”. This enumeration can be represent either a an event or state change (e.g. “transition to Idle”) or a state (e.g. “Idle”) - based on the context in which it is used.

Sensor Specific

An Event/Reading Type Code of 6Fh indicates that the enumeration is defined explicitly for the particular Sensor Type. In an event message, an Event Dir bit of ‘0’ indicates the state has become asserted, while an Event Dir bit of ‘1’ indicates the state has become deasserted. For example, if the Sensor Type Code were 01h, “Processor”, then an Event/Reading Type Code of 6Fh would indicate that the offsets specified in the *Sensor Type Codes* table for “Processor” are to be used. A ‘0’ Event Dir bit with an event offset of 07h (processor presence) would indicate that the event was generated because a processor has become present (inserted). If the Event Type Code were 6Fh, but the Event Dir bit was 1, the same 07h offset would indicate that the event was generated because the processor had become *not present* (removed).

The advantage of the Sensor Specific enumeration is that it provides state and/or event type information that is ‘customized’ to the particular sensor. The disadvantage is that System Management Software requires a-priori knowledge of these enumerations in order to make use of them. Thus, this type of enumeration is used sparingly.

OEM

These Event/Reading Type Codes indicate that the sensor enumeration is OEM defined. It is used in conjunction with a *non-OEM* Sensor Type Code as a way of specifying an OEM-defined enumeration for a standard sensor type. This type of Event/Reading Type Code should be avoided where possible. Making use of it requires a-priori knowledge of the OEM-defined enumeration. Since multiple OEMs could define their own enumerations under the same code, software needs to use the OEM ID (Manufacturer ID) information along with the Event/Reading Type code to correctly interpret the code. The Manufacturer ID value is obtained by issuing a *Get Device ID* command to the controller that generated the event or owns the sensor identifies the OEM.

35.3 SDR Specification of Event Types

The SDR uses an Event/Reading Type Code in the Event/Reading Base Code field to identify the particular enumeration of states or transitions for which the sensor will generate an event. Often, the events that a sensor generates will be only a subset of the enumerated events. Thus, the Event/Reading Base Code is coupled with an 'Event Mask Field' that identifies which elements of the enumeration could actually generate events.

The Assertion Event Mask field is used in the following manner. Suppose the SDR for a sensor has an Event/Reading Base Code field of 02h, "DMI Usage State". According to the Event/Reading Type Code tables, this value indicates that the sensor produces *discrete* events of the generic event enumeration *Transition to Idle, Transition to Active, Transition to Busy*. If the Assertion Event Mask field is 000000000101b this indicates that of those three possible events in the enumeration, the *Transition to Idle* and *Transition to Busy* events may be issued by the sensor, but the *Transition to Active* event will not. The Deassertion Event Mask is used the same way, but to indicate events on the deassertion, rather than assertion, of a state.

35.4 SDR Specification of Reading Types

The SDR has bits that indicate whether the sensor returns a 'present reading' or not. If it does, it also indicates whether the returned reading is discrete or threshold-based, and if an 8-bit analog (multilevel) value is returned.

If the sensor returns an analog reading, additional information such as the raw data format, units, and conversion factors for the reading are specified in other fields of the SDR. System Management Software can use these values for converting the reading from the sensor into units such as volts, degrees centigrade, etc.

If the sensor returns a discrete reading, the Event/Reading Type Code is used for specifying the possible states that can be returned. For these sensors, the Reading Mask field indicates which possible states could be returned from the sensor as a *present reading* in the same manner that the Assertion Event Mask and Deassertion Event Mask fields are used to indicate states that can generate events.

Note that, by convention, the events that a discrete sensor generates must be a subset of the readable states.

35.5 Use of Codes in Event Messages

When a sensor generates an Event Message, the Event/Reading Type Code and corresponding event enumeration 'offset' will be returned in the Event Message. The Event/Reading Type Code is returned in the *Event Type* field, and the enumeration offset value is returned in a bit field in the *Event Data* field. An additional, optional, offset to the 'Severity' enumeration offsets may also be provided if the sensor class is discrete or OEM. In these cases, the base Event/Reading Type Code is implied to be 07h, "DMI-based Severity".

36. Sensor and Event Code Tables

This section contains the tables that define the sensor and event code values used in SDRs and Event Messages.

36.1 Event/Reading Type Codes

Event/Reading Type codes are used in SDRs and Event Messages to indicate the trigger type for an event. These codes are also used in SDRs to indicate what types of *present reading* a sensor provides.

Event/Reading Type Codes are used to specify a particular enumeration that identifies a set of possible events that can be generated by a sensor. For discrete sensors, the specification of an Event/Reading Type code enumeration also indicates the type of reading the sensor provides.

Sensors fall into the following classes:

Sensor Classes:

Discrete Multiple states possible. *Discrete* sensors can contain up to 15 possible states. For discrete sensors, the *Get Sensor Reading* command returns a bit field where each bit reflects a different state. It is possible for a discrete sensor to have more than one state active at a time.

Discrete sensors can be designed to provide either *Generic* or *Sensor-specific* states. The Event/Reading Type Codes in *Table 36-2* are used to specify the particular set of possible *Generic* states for a discrete sensor. Generic states may be applicable to many types of sensors - that is, a temperature sensor and a voltage sensor may both be implemented that return Severity states (Event/Reading Type Code 07h).

For example, Event/Reading Type Code 0Bh indicates a discrete sensor that could return one of three possible states “Redundancy Regained, Redundancy Lost, or Redundancy Degraded”. The offsets from the table correspond to the bit positions in the *Get Sensor Reading* command. The offset values are also used in event messages and in event configuration. (Note, Event Messages reflect only one event at a time. Thus, Event Messages do not return a bit field, just a single offset value corresponding to a single event.)

Sensor-specific states, however, are tied to a particular sensor type. The sensor-specific offsets are specified in the sensor types table: *Table 36-3, Sensor Type Codes*. When the sensor-specific offsets are used in the SDR for a sensor, the Event/Reading Type Code 6Fh is used. This code is also used in Event Messages. An Event Dir bit in the Event Message indicates whether the event is an assertion or deassertion event.

‘Digital’ A digital sensor is not really a unique class, but a term commonly used to refer to special case of a discrete sensor that only has two possible states. Use the discrete sensor type when formatting commands and events for digital sensors. *Table 36-2*, 04h is the Event/Reading code for a ‘digital’ sensor that has the possible generic states “Predictive Failure deasserted” and “Predictive Failure asserted”. If a Get Sensor Reading command returns an offset of ‘0’, then that means the present state is “Predictive Failure deasserted”.

Threshold ‘Threshold based’. Changes event status on reading comparison to threshold values. *Threshold* enumerations may be considered a special case of the discrete sensor type. The Event/Reading Type Code for threshold-based sensors is specified in *Table 36-2, Generic Event/Reading Type Codes*, below. The offsets specify each particular possible threshold state.

Threshold-based sensors return a different response to the *Get Sensor Reading* command than discrete sensors. The offsets The *Get Sensor Reading* command for a threshold-based sensor

contains the present 'analog' reading from the sensor along in addition to the discrete threshold comparison status bit field.

OEM Special case of discrete where the meaning of the states (offsets) are OEM defined.

Table 36-1, Event/Reading Type Code Ranges

Event/Reading Type Code category	7-bit Event/Reading Type Code Range	Sensor Class	Description
unspecified	00h	n/a	Event/Reading Type unspecified.
Threshold	01h	threshold	Threshold-based. Indicates a sensor that utilizes values that represent discrete threshold states in sensor access and/or events. The Event/Reading event offsets for the different threshold states are given in *Table 36-2, Generic Event/Reading Type Codes*, below.
Generic	02h-0Bh	discrete	Generic Discrete. Indicates a sensor that utilizes an Event/Reading Type code & State bit positions / event offsets from one of the sets specified for Discrete or 'digital' Discrete Event/Reading class in *Table 36-2, Generic Event/Reading Type Codes*, below.
Sensor-specific	6Fh	discrete	Sensor-specific Discrete. Indicates that the discrete state information is specific to the sensor type. State bit positions / event offsets for a particular sensor type are specified in the 'sensor-specific offset' column in *Table 36-3, Sensor Type Codes*, below.
OEM	70h-7Fh	OEM	OEM Discrete. Indicates that the discrete state information is specific to the OEM identified by the Manufacturer ID for the IPM device that is providing access to the sensor.

Event/Reading Type Codes that are not explicitly specified in this table are *reserved*.

Table 36-2, Generic Event/Reading Type Codes

Generic Event/Reading Type Code	Event/Reading Class	Generic Offset	Description
			THRESHOLD BASED STATES
01h	Threshold	00h	Lower Non-critical - going low
		01h	Lower Non-critical - going high
		02h	Lower Critical - going low
		03h	Lower Critical - going high
		04h	Lower Non-recoverable - going low
		05h	Lower Non-recoverable - going high
		06h	Upper Non-critical - going low
		07h	Upper Non-critical - going high
		08h	Upper Critical - going low
		09h	Upper Critical - going high
		0Ah	Upper Non-recoverable - going low
		0Bh	Upper Non-recoverable - going high
			DMI-based "Usage State" STATES
02h	Discrete	00h	Transition to Idle
		01h	Transition to Active
		02h	Transition to Busy
	03h		DIGITAL/DISCRETE EVENT STATES
03h	'digital' Discrete	00h	State Deasserted
		01h	State Asserted
04h	'digital' Discrete	00h	Predictive Failure deasserted
		01h	Predictive Failure asserted
05h	'digital' Discrete	00h	Limit Not Exceeded
		01h	Limit Exceeded
06h	'digital' Discrete	00h	Performance Met
		01h	Performance Lags

Generic Event/Reading Type Code	Event/Reading Class	Generic Offset	Description
			SEVERITY EVENT STATES
07h	Discrete	00h	transition to OK
		01h	transition to Non-Critical from OK
		02h	transition to Critical from less severe
		03h	transition to Non-recoverable from less severe
		04h	transition to Non-Critical from more severe
		05h	transition to Critical from Non-recoverable
		06h	transition to Non-recoverable
		07h	Monitor
		08h	Informational
			AVAILABILITY STATUS STATES
08h	'digital' Discrete	00h	Device Removed / Device Absent
		01h	Device Inserted / Device Present
09h	'digital' Discrete	00h	Device Disabled
		01h	Device Enabled
0Ah	Discrete	00h	transition to Running
		01h	transition to In Test
		02h	transition to Power Off
		03h	transition to On Line
		04h	transition to Off Line
		05h	transition to Off Duty
		06h	transition to Degraded
		07h	transition to Power Save
		08h	Install Error
			Other AVAILABILITY STATUS STATES
0Bh	Discrete		<u>Redundancy States</u>
		00h	Fully Redundant (formerly "Redundancy Regained") Indicates that full redundancy has been regained.
		01h	Redundancy Lost Entered any non-redundant state, including Non-redundant:Insufficient Resources.
		02h	Redundancy Degraded Redundancy still exists, but at a less than full level. For example, a system has four fans, and can tolerate the failure of two of them, and presently one has failed.
		03h	Non-redundant:Sufficient Resources from Redundant Redundancy has been lost but unit is functioning with minimum resources needed for 'normal' operation. Entered from Redundancy Degraded or Fully Redundant.
		04h	Non-redundant:Sufficient Resources from Insufficient Resources Unit has regained minimum resources needed for 'normal' operation. Entered from Non-redundant:Insufficient Resources.
		05h	Non-redundant:Insufficient Resources Unit is non-redundant and has insufficient resources to maintain normal operation.
		06h	Redundancy Degraded from Fully Redundant Unit has lost some redundant resource(s) but is still in a redundant state. Entered by a transition from Fully Redundant condition.
		07h	Redundancy Degraded from Non-redundant Unit has regained some resource(s) and is redundant but not fully redundant. Entered from Non-redundant:Sufficient Resources or Non-redundant:Insufficient Resources.
0Ch	Discrete		<u>ACPI Device Power States</u>
		00h	D0 Power State
		01h	D1 Power State
		02h	D2 Power State
		03h	D3 Power State

36.2 Sensor Type Codes and Data

The following table provides the specification of the Sensor Type values and sensor-specific event offsets (if any). Note that generic offsets can be used with any Sensor Type even if the sensor has a definition for sensor-specific offsets. For example, a Processor sensor could be implemented that uses a Generic Event/ReadingType Code of 0Ah to provide a sensor that indicates whether a processor has entered a degraded state (offset 06h for Event/Reading Type Code 0Ah).

Table 36-3, Sensor Type Codes

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
reserved	00h	-	reserved
Temperature	01h	-	Temperature
Voltage	02h	-	Voltage
Current	03h	-	Current
Fan	04h	-	Fan
Physical Security (Chassis Intrusion)	05h	00h	General Chassis Intrusion
		01h	Drive Bay intrusion
		02h	I/O Card area intrusion
		03h	Processor area intrusion
		04h	LAN Leash Lost (system is unplugged from LAN)
		05h	Unauthorized dock/undock
		06h	FAN area intrusion (supports detection of hot plug fan tampering)
Platform Security Violation Attempt	06h	00h	Secure Mode (Front Panel Lockout) Violation attempt
		01h	Pre-boot Password Violation - user password
		02h	Pre-boot Password Violation attempt - setup password
		03h	Pre-boot Password Violation - network boot password
		04h	Other pre-boot Password Violation
		05h	Out-of-band Access Password Violation
Processor	07h	00h	IERR
		01h	Thermal Trip
		02h	FRB1/BIST failure
		03h	FRB2/Hang in POST failure (used hang is believed to be due or related to a processor failure. Use System Firmware Progress sensor for other BIOS hangs.)
		04h	FRB3/Processor Startup/Initialization failure (CPU didn't start)
		05h	Configuration Error
		06h	SM BIOS 'Uncorrectable CPU-complex Error'
		07h	Processor Presence detected
		08h	Processor disabled
		09h	Terminator Presence Detected
Power Supply	08h	00h	Presence detected
		01h	Power Supply Failure detected
		02h	Predictive Failure
		03h	Power Supply AC lost
		04h	AC lost or out-of-range
		05h	AC out-of-range, but present
Power Unit	09h	00h	Power Off / Power Down
		01h	Power Cycle
		02h	240VA Power Down
		03h	Interlock Power Down
		04h	AC lost
		05h	Soft Power Control Failure (unit did not respond to request to turn on)
		06h	Power Unit Failure detected
		07h	Predictive Failure

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
Cooling Device	0Ah	-	-
Other Units-based Sensor (per units given in SDR)	0Bh	-	-
Memory	0Ch	00h 01h 02h 03h 04h 05h	Correctable ECC / other correctable memory error Uncorrectable ECC / other uncorrectable memory error Parity Memory Scrub Failed (stuck bit) Memory Device Disabled Correctable ECC / other correctable memory error logging limit reached *The Event Data 3 field for this command can be used to provide an event extension code, with the following definition:* 7:0 DIMM/SIMM/RIMM identification, relative to the entity that the sensor is associated with (if SDR provided for this sensor)
Drive Slot (Bay)	0Dh	-	-
POST Memory Resize	0Eh	-	-
System Firmware Progress (formerly POST Error)	0Fh	00h	System Firmware Error (POST Error) *The Event Data 2 field can be used to provide an event extension code, with the following definition:* Event Data 2: 00h Unspecified. 01h No system memory is physically installed in the system. 02h No usable system memory, all installed memory has experienced an unrecoverable failure. 03h Unrecoverable hard-disk/ATAPI/IDE device failure. 04h Unrecoverable system-board failure. 05h Unrecoverable diskette subsystem failure. 06h Unrecoverable hard-disk controller failure. 07h Unrecoverable PS/2 or USB keyboard failure. 08h Removable boot media not found 09h Unrecoverable video controller failure 0Ah No video device detected 0Bh Firmware (BIOS) ROM corruption detected 0Ch CPU voltage mismatch (processors that share same supply have mismatched voltage requirements) 0Dh CPU speed matching failure 0Eh to FFh reserved
		01h	System Firmware Hang (uses same Event Data 2 definition as following System Firmware Progress offset)

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
		02h	System Firmware Progress *The Event Data 2 field can be used to provide an event extension code, with the following definition:* Event Data 2: 00h Unspecified. 01h Memory initialization. 02h Hard-disk initialization 03h Secondary processor(s) initialization 04h User authentication 05h User-initiated system setup 06h USB resource configuration 07h PCI resource configuration 08h Option ROM initialization 09h Video initialization 0Ah Cache initialization 0Bh SM Bus initialization 0Ch Keyboard controller initialization 0Dh Embedded controller/management controller initialization 0Eh Docking station attachment 0Fh Enabling docking station 10h Docking station ejection 11h Disabling docking station 12h Calling operating system wake-up vector 13h Starting operating system boot process, e.g. calling Int 19h 14h Baseboard or motherboard initialization 15h reserved 16h Floppy initialization 17h Keyboard test 18h Pointing device test 19h Primary processor initialization 1Ah to FFh reserved
Event Logging Disabled	10h	00h 01h 02h 03h	Correctable Memory Error Logging Disabled Event 'Type' Logging Disabled. Event Logging is disabled for following event/reading type and offset has been disabled. Event Data 2 Event/Reading Type Code Event Data 3 [7:6] - reserved. Write as 00b. [5] 1b = logging has been disabled for all events of given type [4] 1b = assertion event, 0b = deassertion event [3:0] Event Offset Log Area Reset/Cleared All Event Logging Disabled
Watchdog 1	11h	 00h 01h 02h 03h 04h 05h 06h 07h	This sensor is provided to support IPMI v0.9 to v1.0 transition. This is deprecated in IPMI v1.5. See sensor 23h for recommended definition of Watchdog sensor for new v1.0 and for IPMI v1.5 implementations. BIOS Watchdog Reset OS Watchdog Reset OS Watchdog Shut Down OS Watchdog Power Down OS Watchdog Power Cycle OS Watchdog NMI / Diagnostic Interrupt OS Watchdog Expired, status only OS Watchdog pre-timeout Interrupt, non-NMI

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
System Event	12h	00h	System Reconfigured
		01h	OEM System Boot Event
		02h	Undetermined system hardware failure (this event would typically require system-specific diagnostics to determine FRU / failure type)
		03h	Entry added to Auxiliary Log (see *25.12, Get Auxiliary Log Status* Command and *25.13, Set Auxiliary Log Status* Command, for more information) Event Data 2 [7:4] - Log Entry Action 0h = entry added 1h = entry added because event did not be map to standard IPMI event 2h = entry added along with one or more corresponding SEL entries 3h = log cleared 4h = log disabled 5h = log enabled all other = reserved [3:0] - Log Type 0h = MCA Log 1h = OEM 1 2h = OEM 2 all other = reserved
		04h	PEF Action Event Data 2 The following bits reflect the PEF Actions that are about to be taken after the event filters have been matched. The event is captured *before* the actions are taken. [7:6] - reserved [5] - 1b = Diagnostic Interrupt (NMI) [4] - 1b = OEM action [3] - 1b = power cycle [2] - 1b = reset [1] - 1b = power off [0] - 1b = Alert
Critical Interrupt	13h	00h	Front Panel NMI / Diagnostic Interrupt
		01h	Bus Timeout
		02h	I/O channel check NMI
		03h	Software NMI
		04h	PCI PERR
		05h	PCI SERR
		06h	EISA Fail Safe Timeout
		07h	Bus Correctable Error
		08h	Bus Uncorrectable Error
		09h	Fatal NMI (port 61h, bit 7)
Button	14h	00h	Power Button pressed
		01h	Sleep Button pressed
		02h	Reset Button pressed
Module / Board	15h	-	-
Microcontroller / Coprocessor	16h	-	-
Add-in Card	17h	-	-
Chassis	18h	-	-
Chip Set	19h	-	-
Other FRU	1Ah	-	-
Cable / Interconnect	1Bh	-	-
Terminator	1Ch	-	-

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
System Boot Initiated	1Dh	00h	Initiated by power up
		01h	Initiated by hard reset
		02h	Initiated by warm reset
		03h	User requested PXE boot
		04h	Automatic boot to diagnostic
Boot Error	1Eh	00h	No bootable media
		01h	Non-bootable diskette left in drive
		02h	PXE Server not found
		03h	Invalid boot sector
		04h	Timeout waiting for user selection of boot source
OS Boot	1Fh	00h	A: boot completed
		01h	C: boot completed
		02h	PXE boot completed
		03h	Diagnostic boot completed
		04h	CD-ROM boot completed
		05h	ROM boot completed
		06h	boot completed - boot device not specified
OS Critical Stop	20h	00h	Stop during OS load / initialization
		01h	Run-time Stop
Slot / Connector	21h	00h	Fault Status asserted
		01h	Identify Status asserted
		02h	Slot / Connector Device installed/attached [This can include dock events]
		03h	Slot / Connector Ready for Device Installation - Typically, this means that the slot power is off. The Ready for Installation, Ready for Removal, and Slot Power states can transition together, depending on the slot implementation.
		04h	Slot/Connector Ready for Device Removal
		05h	Slot Power is Off
		06h	Slot / Connector Device Removal Request - This is typically connected to a switch that becomes asserted to request removal of the device)
		07h	Interlock asserted - This is typically connected to a switch that mechanically enables/disables power to the slot, or locks the slot in the 'Ready for Installation / Ready for Removal states' - depending on the slot implementation. The asserted state indicates that the lock-out is active.
		08h	Slot is Disabled *The Event Data 2 & 3 fields can be used to provide an event extension code, with the following definition:* <u>Event Data 2</u> 7 reserved 6:0 Slot/Connector Type 0 PCI 1 Drive Array 2 External Peripheral Connector 3 Docking 4 other standard internal expansion slot 5 slot associated with entity specified by Entity ID for sensor all other = reserved <u>Event Data 3</u> 7:0 Slot/Connector Number

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
System ACPI Power State	22h	00h	S0 / G0 “working”
		01h	S1 “sleeping with system h/w & processor context maintained”
		02h	S2 “sleeping, processor context lost”
		03h	S3 “sleeping, processor & h/w context lost, memory retained.”
		04h	S4 “non-volatile sleep / suspend-to disk”
		05h	S5 / G2 “soft-off”
		06h	S4 / S5 soft-off, particular S4 / S5 state cannot be determined
		07h	G3 / Mechanical Off
		08h	Sleeping in an S1, S2, or S3 states (used when particular S1, S2, S3 state cannot be determined)
		09h	G1 sleeping (S1-S4 state cannot be determined)
		0Ah	S5 entered by override
		0Bh	Legacy ON state
		0Ch	Legacy OFF state
		0Eh	Unknown
Watchdog 2	23h		This sensor is recommended for new IPMI v1.0 and later implementations.
		00h	Timer expired, status only (no action, no interrupt)
		01h	Hard Reset
		02h	Power Down
		03h	Power Cycle
		04h-07h	reserved
		08h	Timer interrupt The Event Data 2 field for this command can be used to provide an event extension code, with the following definition: 7:4 interrupt type 0h = none 1h = SMI 2h = NMI 3h = Messaging Interrupt Fh = unspecified all other = reserved 3:0 timer use at expiration: 0h = reserved 1h = BIOS FRB2 2h = BIOS/POST 3h = OS Load 4h = SMS/OS 5h = OEM Fh = unspecified all other = reserved
Platform Alert	24h		This sensor can be used for returning the state and generating events associated with alerts that have been generated by the platform mgmt. subsystem
		00h	platform generated page
		01h	platform generated LAN alert
		02h	Platform Event Trap generated, formatted per IPMI PET specification
		03h	platform generated SNMP trap, OEM format
Entity Presence	25h		This sensor type provides a mechanism that allows a management controller to direct system management software to ignore a set of sensors based on detecting that presence of an entity. This sensor type is not typically used for event generation - but to just provide a present reading.
		00h	Entity Present. This indicates that the Entity identified by the Entity ID for the sensor is present.
		01h	Entity Absent. This indicates that the Entity identified by the Entity ID for the sensor is absent. If the entity is absent, system management software should consider all sensors associated with that Entity to be absent as well - and ignore those sensors.
		02h	Entity Disabled. The Entity is present, but has been disabled. A deassertion of this event indicates that the Entity has been enabled.
Monitor ASIC / IC	26h	-	-

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
LAN	27h	00h 01h	LAN Heartbeat Lost LAN Heartbeat
Management Subsystem Health	28h	00h 01h 02h 03h	sensor access degraded or unavailable controller access degraded or unavailable management controller off-line management controller unavailable
Battery	29h	00h 01h 02h	battery low (predictive failure) battery failed battery presence detected
Reserved	remaining	-	-
OEM RESERVED	C0h-FFh	-	-

37. Sensor Data Record Formats

The general Sensor Data Record format consists of three major components, the Record Header, Record 'Key' fields, and the Record Body. In order to save space, Sensor Data Records are not required for sensors that only generate events. In particular, sensors that do not require initialization by the initialization agent, and are not expected to be accessed by system management software to obtain present readings, etc. (Generic system management software does not access sensors that are not reported via the SDRs).

RECORD HEADER

Is the same for all records, consisting of:

Record ID: A value that's used for accessing Sensor Data Records. Note that since Record IDs may be reassigned, retrievers of a record should use the 'KEY' fields to verify that the expected record was obtained.

SDR Version: The version number of the SDR specification. Used in conjunction with Record Type, following.

Record Type: A number representing the type of the record. E.g. 01h = *8-bit Sensor with Thresholds.*

Record Length: Number of bytes of data following the Record Length field.

RECORD 'KEY' FIELDS

The Record 'Key' Fields are a set of fields that together are unique amongst instances of a given record type. For example, for 'sensor' records, these fields specify the location (e.g. slave address, LUN, and Bus ID) and sensor number of the sensor. The Record Key bytes shall be contiguous and follow the Record Header. The number of bytes that make up the Record Key field may vary according to record type.

RECORD BODY

The remaining Record Type specific information for the particular sensor data record.

37.1 SDR Type 01h, Full Sensor Record

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

Sensor Capability	Supported in Full Sensor Record? (type 01h)	Supported in Compact Sensor Record? (type 02h)
Sensor provides analog readings	yes	no
Sensor requires threshold value initialization	yes	no
Sensor requires hysteresis value initialization	yes	yes*
Multiple sensors can share record	no	yes*

* Note: sensors that require unique event or hysteresis configuration cannot share a record.

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

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

Table 37-1, Full Sensor Record - SDR Type 01h

byte	Field Name	size	Description
	SENSOR RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. It is *not* related to the sensor ID.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.*
4	Record Type	1	Record Type Number = 01h, Full Sensor Record
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Sensor Owner ID	1	[7:1] - 7-bit I^2C Slave Address, or 7-bit system software ID[2] [0] - 0b = ID is IPMB Slave Address, 1b = system software ID
7	Sensor Owner LUN	1	[7:4] - Channel Number The Channel Number can be used to specify access to sensors that are located on management controllers that are connected to the BMC via channels other than the primary IPMB. (Note: In IPMI v1.5 the ordering of bits 7:2 of this byte have changed to support the 4-bit channel number.) [3:2] - reserved [1:0] - Sensor Owner LUN. LUN in the Sensor Owner that is used to send/receive IPMB messages to access the sensor. 00b if system software is Sensor Owner.
8	Sensor Number	1	Unique number identifying the sensor behind a given slave address and LUN. Code FFh reserved.
	RECORD BODY BYTES		
9	Entity ID	1	Indicates the physical entity that the sensor is monitoring or is otherwise associated with the sensor. See Table 37-12, Entity ID Codes.

byte	Field Name	size	Description
10	Entity Instance	1	[7] - 0b = treat entity as a physical entity per Entity ID table 1b = treat entity as a logical container entity. For example, if this bit is set, and the Entity ID is 'Processor', the container entity would be considered to represent a logical 'Processor Group' rather than a physical processor. This bit is typically used in conjunction with an Entity Association record. [6:0] - Instance number for entity. (See section *33.1, System- and Device-relative Entity Instance* Values for more information) 00h-5Fh system-relative Entity Instance. The Entity Instance number must be unique for each different entity of the same type Entity ID in the system. 60h-7Fh device-relative Entity Instance. The Entity Instance number must only be unique relative to the management controller providing access to the Entity.
11	Sensor Initialization	1	[7] - reserved. Write as 0b. [6] - Init Scanning 1b = enable scanning (this bit=1 implies that the sensor accepts the 'enable/disable scanning' bit in the *Set Sensor Event Enable* command). [5] - Init Events 1b = enable events (per Sensor Event Message Control Support bits in Sensor Capabilities field, and per the Event Mask fields, below). [4] - Init Thresholds 1b = initialize sensor thresholds (per settable threshold mask below). [3] - Init Hysteresis 1b = initialize sensor hysteresis (per Sensor Hysteresis Support bits in the Sensor Capabilities field, below). [2] - Init Sensor Type 1b = initialize Sensor Type and Event / Reading Type code Sensor Default (power up) State Reports how this sensor comes up on device power up and hardware/cold reset. The Initialization Agent does not use this bit. This bit solely reports to software how the sensor comes prior to being initialized by the Initialization Agent. [1] - 0b = event generation disabled, 1b = event generation enabled [0] - 0b = sensor scanning disabled, 1b = sensor scanning enabled

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

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

byte	Field Name	size	Description
17 18	Deassertion Event Mask / Upper Threshold Reading Mask	2	Deassertion Event Mask (for non- threshold-based sensors) The Event Mask bytes are a bit mask that specifies support for 15 successive events starting with the event specified by Event/Reading Type Code. LS byte first. [15] - reserved. Write as 0b [14:0] - Event offsets 14 through 0, respectively. 1b = assertion event can be generated for this state. Upper Threshold Reading Mask (for threshold-based sensors) Indicates which upper threshold comparison status is returned via the *Get Sensor Reading* command. [15] - reserved. Write as 0b [14] - 1b = Upper non-recoverable threshold comparison is returned [13] - 1b = Upper critical threshold is comparison returned [12] - 1b = Upper non-critical threshold is comparison returned Threshold Deassertion Event Mask [11] - 1b = deassertion event for upper non-recoverable going high supported [10] - 1b = deassertion event for upper non-recoverable going low supported [9] - 1b = deassertion event for upper critical going high supported [8] - 1b = deassertion event for upper critical going low supported [7] - 1b = deassertion event for upper non-critical going high supported [6] - 1b = deassertion event for upper non-critical going low supported [5] - 1b = deassertion event for lower non-recoverable going high supported [4] - 1b = deassertion event for lower non-recoverable going low supported [3] - 1b = deassertion event for lower critical going high supported [2] - 1b = deassertion event for lower critical going low supported [1] - 1b = deassertion event for lower non-critical going high supported [0] - 1b = deassertion event for lower non-critical going low supported
19 20	Discrete Reading Mask / Settable Threshold Mask, Readable Threshold Mask	2	Reading Mask (for non- threshold based sensors) Indicates what discrete readings can be returned by this sensor, or, for threshold based sensors, this indicates which thresholds are settable and which are readable. The Reading Mask bytes are a bit mask that specifies support for 15 successive states starting with the value from *Table 36-1, Event/Reading Type Code Ranges*. LS byte first. [15] - reserved. Write as 0b [14:0] - state bits 0 through 14. 1b = discrete state can be returned by this sensor. Settable Threshold Mask (for threshold-based sensors) Indicates which thresholds are settable via the *Set Sensor Thresholds*. This mask also indicates which threshold values will be initialized if the 'Init Events' bit is set. LS byte first. [15:14] - reserved. Write as 00b. [13] - 1b = Upper non-recoverable threshold is settable [12] - 1b = Upper critical threshold is settable [11] - 1b = Upper non-critical threshold is settable [10] - 1b = Lower non-recoverable threshold is settable [9] - 1b = Lower critical threshold is settable [8] - 1b = Lower non-critical threshold is settable Readable Threshold Mask (for threshold-based sensors) Indicates which thresholds are readable via the *Get Sensor Thresholds* command. [7:6] - reserved. Write as 00b. [5] - 1b = Upper non-recoverable threshold is readable [4] - 1b = Upper critical threshold is readable [3] - 1b = Upper non-critical threshold is readable [2] - 1b = Lower non-recoverable threshold is readable [1] - 1b = Lower critical threshold is readable [0] - 1b = Lower non-critical threshold is readable

byte	Field Name	size	Description
21	Sensor Units 1	1	[7:6] - Analog (numeric) Data Format** 00b = unsigned 01b = 1's complement (signed) 10b = 2's complement (signed) 11b = Does not return analog (numeric) reading [5:3] - Rate unit 000b = none 001b = per µS 010b = per ms 011b = per s 100b = per minute 101b = per hour 110b = per day 111b = reserved [2:1] - Modifier unit 00b = none 01b = Basic Unit / Modifier Unit 10b = Basic Unit * Modifier Unit 11b = reserved [0] - Percentage 0b = no, 1b = yes ** Specifies threshold and 'analog' reading, if 'analog' reading provided. If neither thresholds nor analog reading are provided, this field should be written as 00h.
22	Sensor Units 2 - Base Unit	1	[7:0] - Units Type code: See *Table 37-14, Sensor Unit Type Codes*.
23	Sensor Units 3 - Modifier Unit	1	[7:0] - Units Type code, 00h if unused.
24	Linearization	1	[7] - reserved [6:0] - enum (linear, ln, log10, log2, e, exp10, exp2, 1/x, sqr(x), cube(x), sqrt(x), $cube^{-1}$ (x)) - 70h = non-linear. 71h-7Fh = non-linear, OEM defined.
25	M	1	[7:0] - M: LS 8 bits [2's complement, signed, 10 bit 'M' value.] -
26	M, Tolerance	1	[7:6] - M: MS 2 bits [5:0] - Tolerance: 6 bits, unsigned (Tolerance in +/- ½ raw counts)
27	B	1	[7:0] - B: LS 8 bits [2's complement, signed, 10-bit 'B' value.] -
28	B, Accuracy	1	[7:6] - B: MS 2 bits Unsigned, 10-bit Basic Sensor Accuracy in 1/100 percent scaled up by unsigned Accuracy exponent: [5:0] - Accuracy: LS 6 bits
29	Accuracy, Accuracy exp	1	[7:4] - Accuracy: MS 4 bits [3:2] - Accuracy exp: 2 bits, unsigned [1:0] - reserved: 2 bits
30	R exp, B exp	1	[7:4] - R (result) exponent 4 bits, 2's complement, signed [3:0] - B exponent 4 bits, 2's complement, signed
31	Analog characteristic flags	1	[7:3] - reserved [2] - normal min specified 1b = yes, 0b = normal min field unspecified [1] - normal max specified 1b = yes, 0b = normal max field unspecified [0] - nominal reading specified 1b = yes, 0b = nominal reading field unspecified
32	Nominal Reading	1	Given as a raw value. Must be converted to units-based value using the 'y=Mx+B' formula. 1's or 2's complement signed or unsigned per flag bits in Sensor Units 1.
33	Normal Maximum	1	Given as a raw value. Must be converted to units-based value using the 'y=Mx+B' formula. 1's or 2's complement signed or unsigned per 'signed' bit in Sensor Units 1.
34	Normal Minimum	1	Given as a raw value. Must be converted to units-based value using the 'y=Mx+B' formula. Signed or unsigned per 'signed' bit in Sensor Units 1.
35	Sensor Maximum Reading	1	Given as a raw value. Must be converted to units-based value based using the y=Mx+B formula. Signed or unsigned per 'signed' bit in sensor flags. Normally 'FFh' for an 8-bit unsigned sensor, but can be a lesser value if the sensor has a restricted range. If max. reading cannot be pre-specified this value should be set to max value, based on data format, (e.g. FFh for an unsigned sensor, 7Fh for 2's complement, etc.)
36	Sensor Minimum Reading	1	Given as a raw value. Must be converted to units-based value using the 'y=Mx+B' formula. Signed or unsigned per 'signed' bit in sensor flags. If min. reading cannot be pre-specified this value should be set to min value, based on data format, (e.g. 00h for an unsigned sensor, 80h for 2's complement, etc.)

byte	Field Name	size	Description
37	Upper non-recoverable Threshold	1	Use of this field is based on Settable Threshold Mask. If the corresponding bit is set in the mask byte and the 'Init Sensor Thresholds' bit is also set, then this value will be used for initializing the sensor threshold. Otherwise, this value should be ignored. The thresholds are given as raw values that must be converted to units-based values using the 'y=Mx+B' formula.
38	Upper critical Threshold	1	Use of this field is based on Settable Threshold Mask, above
39	Upper non-critical Threshold	1	Use of this field is based on Settable Threshold Mask, above
40	Lower non-recoverable Threshold	1	Use of this field is based on Settable Threshold Mask, above
41	Lower critical Threshold	1	Use of this field is based on Settable Threshold Mask, above
42	Lower non-critical Threshold	1	Use of this field is based on Settable Threshold Mask, above
43	Positive-going Threshold Hysteresis value	1	Positive hysteresis is defined as the unsigned number of counts that are subtracted from the raw threshold values to create the 're-arm' point for all positive-going thresholds on the sensor. 0 indicates that there is no hysteresis on positive-going thresholds for this sensor. Hysteresis values are given as raw counts. That is, to find the degree of hysteresis in units, the value must be converted using the 'y=Mx+B' formula.
44	Negative-going Threshold Hysteresis value	1	Negative hysteresis is defined as the unsigned number of counts that are added to the raw threshold value to create the 're-arm' point for all negative-going thresholds on the sensor. 0 indicates that there is no hysteresis on negative-going thresholds for this sensor.
45	reserved	1	reserved. Write as 00h.
46	reserved	1	reserved. Write as 00h.
47	OEM	1	Reserved for OEM use.
48	ID String Type/Length Code	1	Sensor 'ID' String Type/Length Code, per Section *37.14, Type/Length Byte Format*.
49: +N	ID String Bytes	N	Sensor ID String bytes. Only present if non-zero length in Type/Length field. *16 bytes, maximum. Note: the SDR can be implemented as a fixed length record. Bytes beyond the ID string bytes are unspecified and should be ignored.*

Notes:

1. Resolution, as used in the DMI Systems Standard Groups 'probes' groups, can be obtained from the "m" factor in the 'y=mx+b' reading conversion.
2. 7-bit I^2C Slave Address field. By convention, we normally designate an I^2C slave address as an eight-bit number with the least-significant bit always 0. E.g. 20h = 00100000b. The 7-bit Slave Address field holds the most-significant 7 bits of this value. E.g. 0010000b.

37.2 SDR Type 02h, Compact Sensor Record

The Full Sensor Record can be used to describe any type of sensor. The *Compact* sensor record saves space, but has limitations in the sensors it can describe. The Full record is defined as a 64-byte record, while the Compact record is defined as 48-bytes. See previous section for a description of other differences between the Full Sensor Record and the Compact Sensor Record.

Table 37-2, Compact Sensor Record - SDR Type 02h

byte	Field Name	size	Description
	SENSOR RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. It is *not* related to the sensor ID.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.*
4	Record Type	1	Record Type Number = 02h, Compact Sensor Record
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Sensor Owner ID	1	[7:1] - 7-bit I^2C Slave Address, or 7-bit system software ID[1] [0] - 0b = ID is IPMB Slave Address, 1b = system software ID
7	Sensor Owner LUN	1	[7:4] - Channel Number The Channel Number can be used to specify access to sensors that are located on management controllers that are connected to the BMC via channels other than the primary IPMB. (Note: In IPMI v1.5 the ordering of bits 7:2 of this byte have changed to support the 4-bit channel number) [3:2] - FRU Inventory Device Owner LUN. LUN for Write/Read FRU commands to access FRU information. 00b otherwise. [1:0] - Sensor Owner LUN. LUN in the Sensor Owner that is used to send/receive IPMB messages to access the sensor. 00b if system software is Sensor Owner.
8	Sensor Number	1	Unique number identifying the sensor behind the given slave address and LUN. Code FFh reserved.
	RECORD BODY BYTES		
9	Entity ID	1	Indicates the physical entity that the sensor is monitoring or is otherwise associated. See *Table 37-12, Entity ID Codes*.
10	Entity Instance	1	[7] - 0b = treat entity as a physical entity per Entity ID table 1b = treat entity as a logical container entity. For example, if this bit is set, and the Entity ID is 'Processor', the container entity would be considered to represent a logical 'Processor Group' rather than a physical processor. This bit is typically used in conjunction with an Entity Association record. [6:0] - Instance number for entity. (See section *33.1, System- and Device-relative Entity Instance* Values for more information) 00h-5Fh system-relative Entity Instance. The Entity Instance number must be unique for each different entity of the same type Entity ID in the system. 60h-7Fh device-relative Entity Instance. The Entity Instance number must only be unique relative to the management controller providing access to the Entity.

byte	Field Name	size	Description
11	Sensor Initialization	1	[7] - reserved. Write as 0b. [6] - Init Scanning 1b = enable scanning (this bit=1b implies that the sensor accepts the 'enable/disable scanning' bit in the *Set Sensor Event Enable* command). [5] - Init Events 1b = enable events (per Sensor Event Message Control Support bits in Sensor Capabilities field, and per the Event Mask fields, below.) [4] - reserved. Write as 0b. [3] - Init Hysteresis 1b = initialize sensor hysteresis (per Sensor Hysteresis Support bits in the Sensor Capabilities field, below). [2] - Init Sensor Type 1b = initialize Sensor Type and Event / Reading Type code <u>Sensor Default (power up) State</u> Reports how this sensor comes up on device power up and hardware/cold reset. The Initialization Agent does not use this bit. This bit solely reports to software how the sensor comes prior to being initialized by the Initialization Agent. [1] - 0b = event generation disabled, 1b = event generation enabled [0] - 0b = sensor scanning disabled, 1b = sensor scanning enabled
12	Sensor Capabilities	1	[7] - 1b = ignore sensor if Entity is not present or disabled. 0b = don't ignore sensor. <u>Sensor Auto Re-arm Support</u> Indicates whether the sensor requires manual rearming, or automatically rearms itself when the event clears. 'manual' implies that the *get sensor event status* and *rearm sensor events* commands are supported [6] - 0b = no (manual), 1b = yes (auto) <u>Sensor Hysteresis Support</u> [5:4] - 00b = No hysteresis, or hysteresis built-in but not specified. 01b = hysteresis is readable. 10b = hysteresis is readable and settable. 11b = Fixed, unreadable, hysteresis. Hysteresis fields values implemented in the sensor. <u>Sensor Threshold Access Support</u> [3:2] - 00b = no thresholds. 01b = thresholds are readable, per Reading Mask, below. 10b = reserved 11b = Fixed, unreadable, thresholds. Which thresholds are supported is reflected by the Reading Mask. The threshold value fields report the values that are 'hard-coded' in the sensor. <u>Sensor Event Message Control Support</u> Indicates whether this sensor generates Event Messages, and if so, what type of Event Message control is offered. [1:0] - 00b = per threshold/discrete-state event enable/disable control (implies that entire sensor and global disable are also supported) 01b = entire sensor only (implies that global disable is also supported) 10b = global disable only 11b = no events from sensor
13	Sensor Type	1	Code representing the sensor type. From the Table 36-3, Sensor Type Codes. E.g. Temperature, Voltage, Processor, etc.
14	Event / Reading Type Code	1	Event/Reading Type Code. From the Table 36-1, Event/Reading Type Code Ranges.

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

byte	Field Name	size	Description
17 18	Deassertion Event Mask / Upper Threshold Reading Mask	2	Deassertion Event Mask (for non- threshold-based sensors) The Event Mask bytes are a bit mask that specifies support for 15 successive events starting with the event specified by Event/Reading Type Code. LS byte first. [15] - reserved. Write as 0b [14:0] - Event offsets 14 through 0, respectively. 1b = assertion event can be generated for this state. Upper Threshold Reading Mask (for threshold-based sensors) Indicates which upper threshold comparison status is returned via the *Get Sensor Reading* command. [15] - reserved. Write as 0b [14] - Upper non-recoverable threshold comparison is returned [13] - Upper critical threshold is comparison returned [12] - Upper non-critical threshold is comparison returned Threshold Deassertion Event Mask [11] - 1b = deassertion event for upper non-recoverable going high supported [10] - 1b = deassertion event for upper non-recoverable going low supported [9] - 1b = deassertion event for upper critical going high supported [8] - 1b = deassertion event for upper critical going low supported [7] - 1b = deassertion event for upper non-critical going high supported [6] - 1b = deassertion event for upper non-critical going low supported [5] - 1b = deassertion event for lower non-recoverable going high supported [4] - 1b = deassertion event for lower non-recoverable going low supported [3] - 1b = deassertion event for lower critical going high supported [2] - 1b = deassertion event for lower critical going low supported [1] - 1b = deassertion event for lower non-critical going high supported [0] - 1b = deassertion event for lower non-critical going low supported
19 20	Discrete Reading Mask / Settable Threshold Mask, Readable Threshold Mask	2	Reading Mask (for non- threshold based sensors) Indicates what discrete readings can be returned by this sensor, or, for threshold based sensors, this indicates which thresholds are settable and which are readable. The Reading Mask bytes are a bit mask that specifies support for 15 successive states starting with the value from *Table 36-1, Event/Reading Type Code Ranges*. LS byte first. [15] - reserved. Write as 0b [14:0] - state bits 0 through 14. 1b = discrete state can be returned by this sensor. Settable Threshold Mask (for threshold-based sensors) Indicates which thresholds are settable via the *Set Sensor Thresholds*. This mask also indicates which threshold values will be initialized if the 'Init Events' bit is set. LS byte first. [15:14] - reserved. Write as 00b. [13] - Upper non-recoverable threshold is settable [12] - Upper critical threshold is settable [11] - Upper non-critical threshold is settable [10] - Lower non-recoverable threshold is settable [9] - Lower critical threshold is settable [8] - Lower non-critical threshold is settable Readable Threshold Mask (for threshold-based sensors) Indicates which thresholds are readable via the *Get Sensor Thresholds* command. [7:6] - reserved. Write as 00b. [5] - Upper non-recoverable threshold is readable [4] - Upper critical threshold is readable [3] - Upper non-critical threshold is readable [2] - Lower non-recoverable threshold is readable [1] - Lower critical threshold is readable [0] - Lower non-critical threshold is readable

byte	Field Name	size	Description
21	Sensor Units 1	1	[7:6] - reserved. Write as 11b. [5:3] - Rate unit 000b = none 001b = per µS 010b = per ms 011b = per s 100b = per minute 101b = per hour 110b = per day 111b = reserved [2:1] - Modifier unit 00b = none 01b = Basic Unit / Modifier Unit 10b = Basic Unit * Modifier Unit 11b = reserved [0] - Percentage 0b = no, 1b = yes
22	Sensor Units 2 - Base Unit	1	[7:0] - Units Type code: See Units Type Codes table.
23	Sensor Units 3 - Modifier Unit	1	[7:0] - Units Type code, 00h if unused.
24	Sensor Record Sharing	2	Byte 1: [7:6] - reserved ID String Instance Modifier Type (The instance modifier is a character(s) that software can append to the end of the ID String. This field selects whether the appended character(s) will be numeric or alpha. The Instance Modified Offset field, below, selects the starting value for the character.) [5:4] - 00b = numeric 01b = alpha Share Count [3:0] - Share count (number of sensors sharing this record). Sensor numbers sharing this record are sequential starting with the sensor number specified by the *Sensor Number* field for this record. E.g. if the starting sensor number was 10, and the share count was 3, then sensors 10, 11, and 12 would share this record. Byte 2: Entity Instance Sharing [7] - 0b = Entity Instance same for all shared records 1b = Entity Instance increments for each shared record [6:0] - ID String Instance Modifier Offset Multiple Discrete sensors can share the same sensor data record. The ID String Instance Modifier and Modifier Offset are used to modify the Sensor ID String as follows: Suppose sensor ID is “Temp ” for ‘Temperature Sensor’, share count = 3, ID string instance modifier = numeric, instance modifier offset = 5 - then the sensors could be identified as: Temp 5, Temp 6, Temp 7 If the modifier = alpha, offset=0 corresponds to ‘A’, offset=25 corresponds to ‘Z’, and offset = 26 corresponds to ‘AA’, thus, for offset=26 the sensors could be identified as: Temp AA, Temp AB, Temp AC (alpha characters are considered to be base 26 for ASCII)
26	Positive-going Threshold Hysteresis value	1	Positive hysteresis is defined as the unsigned number of counts that are subtracted from the raw threshold values to create the ‘re-arm’ point for all positive-going thresholds on the sensor. 0 indicates that there is no hysteresis on positive-going thresholds for this sensor. Hysteresis values are given as raw counts. That is, to find the degree of hysteresis in units, the value must be converted using the ‘y=Mx+B’ formula. *Note: Cannot use shared record if sensors require individual hysteresis settings.*

byte	Field Name	size	Description
27	Negative-going Threshold Hysteresis value	1	Negative hysteresis is defined as the unsigned number of counts that are added to the raw threshold value to create the 're-arm' point for all negative-going thresholds on the sensor. 0 indicates that there is no hysteresis on negative-going thresholds for this sensor. *Note: Cannot use shared record if sensors require individual hysteresis settings.*
28	reserved	1	reserved. Write as 00h.
29	reserved	1	reserved. Write as 00h.
30	reserved	1	reserved. Write as 00h.
31	OEM	1	Reserved for OEM use.
32	ID String Type/Length Code	1	Sensor 'ID' String Type/Length Code, per Section *37.14, Type/Length Byte Format*.
33: +N	ID String Bytes	N	Sensor ID String bytes. Only present if non-zero length in Type/Length field. *16 bytes, maximum.*

Notes:

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

37.3 SDR Type 08h - Entity Association Record

This record is used to present the relationship between entities that contain, or are contained by, other entities. For example, a particular Power Unit entity may consist of multiple Power Supply entities. In this case, the particular Power Unit is designated as the *container* entity, while the individual Power Supply entities are designated as *contained* entities. System Management Software can use this information to recognize the relationship between 'logical' entities (e.g. Cooling Unit), which typically would not have FRU information, and the physical FRUs that comprise that unit (e.g. Fans). A 'Group' Entity ID is provided to be used as the container Entity if the container is not covered by a pre-defined Entity.

System Management Software can use the Entity Association information to correlate events or sensor information between the sensors for the container entity, and the contained entity. For example, a Power Unit may have a redundancy sensor associated with it, while the individual Power Supplies within that Power Unit may have failure status sensors. In the case of a Power Supply failure, there could be two events generated - one for the Power Supply failure, and another for a loss of redundancy in the Power Unit. Having an Entity Association record allows System Management Software to determine the inter-relationship between these events.

Each Entity Association Record can represent contained entities as a four entry list, or as up to two ranges of entity instances. For example, suppose you have four Power Supply entities, with Instance IDs 1, 2, 7, and 8. This could be listed as four separate entity/entity instance pairs, or as two ranges (range 1 comprised of entity instances 1 through 2, and range 2 comprised of entity instances 7 through 8).

Entity Association records can be 'linked' if necessary to extend the number of contained entities under a given container entity. This would be needed if there were more than four contained entities of different entity types, or with non-sequential instance IDs. Entity Association records that have the same *Container Entity* value and a non-zero *Record Link* bit are considered to be linked. The contained entities from each linked record combine to form the complete set of contained entities under the specified container entity.

The *Container Record Link* bit informs system software there is intentionally more than one Entity Association record with the same *Container Entity* value. It is considered an error if Entity Association records are detected that share the same *Container Entity* value but do not have the *Record Link* bit set. In addition, Entity/Entity Instance ID pairs should not be repeated within an Entity Association record, nor between linked Entity Association records.

This record only supports associations between entities that have system-relative Entity Instance values. See section *33.1, System- and Device-relative Entity Instance Values* for more information

Table 37-3, Entity Association Record - SDR Type 08h

byte	Field Name	size	Description
	RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. This may not actually be stored, but may be calculated when records are accessed.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *This is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits. E.g. 51h corresponds to "1.5".*
4	Record Type	1	Record Type Number = 08h, Entity Association. The "RECORD KEY BYTES" includes the first contained entity in order to allow records to be linked according to the Container entity ID. Thus, more than one Entity Association Record can have the same values for bytes 6, 7, and 8 - but must differ in bytes 9 & 10.
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Container Entity ID	1	Entity ID for container entity
7	Container Entity Instance	1	Instance ID for container entity
8	flags	1	[7] - 0b = contained entities specified as list 1b = contained entities specified as range [6] - Record Link 0b = no linked Entity Association records 1b = linked Entity Association records exist [5] - 0b = Container entity and contained entities can be assumed absent if presence sensor for container entity cannot be accessed. This value is also used if the entity does not have a presence sensor. 1b = Presence sensor should always be accessible. Software should consider it an error if the presence sensor associated with the container entity is not accessible. If a presence sensor is accessible, then the presence sensor can still report that the container entity is absent. [4:0] - reserved, write as 00000b
9	Contained Entity 1 / Range 1 entity	1	If list: Entity ID for contained entity 1 If range: Entity ID of entity for contained entity range 1
10	Contained Entity 1 Instance Range 1 first entity instance	1	If list: Instance ID for contained entity 1 If range: Instance ID for first entity in contained entity range 1
	RECORD BODY BYTES		
11	Contained Entity 2 / Range 1 entity	1	00h = unspecified If list: Entity ID for contained entity 2 If range: Entity ID of entity for contained entity range 1 (must match byte 9)
12	Contained Entity 2 Instance / Range 1 last entity Instance	1	00h = unspecified If list: Instance ID for contained entity 2 If range: Instance ID for last entity in contained entity range 1
13	Contained Entity 3 / Range 2 entity	1	00h = unspecified If list: Entity ID for contained entity 3 If range: Entity ID of entity for contained entity range 2
14	Contained Entity 3 Instance / Range 2 first entity Instance	1	00h = unspecified If list: Instance ID for contained entity 3 If range: Instance ID for first entity in contained entity range 2
15	Contained Entity 4 / Range 2 entity	1	00h = unspecified If list: Entity ID for contained entity 4 If range: Entity ID for entity for contained entity range 2 (must match byte 13)
16	Contained Entity 4 Instance / Range 2 last entity Instance	1	00h = unspecified If list: Instance ID for contained entity 4 If range: Instance ID for last entity in contained entity range 2

37.4 SDR Type 09h - Device-relative Entity Association Record

This record is the same as the Type 08h Entity Association record, except that it supports describing associations between entities that have device-relative Entity Instance values as well as system-relative values. See section *33.1, System- and Device-relative Entity Instance Values* and the description for SDR Type 08h for more information.

Table 37-4, Device-relative Entity Association Record - SDR Type 09h

byte	Field Name	size	Description
	RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. This may not actually be stored, but may be calculated when records are accessed.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *This is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits. E.g. 51h corresponds to "1.5".*
4	Record Type	1	Record Type Number = 09h, Device-relative Entity Association. The "RECORD KEY BYTES" includes the first contained entity in order to allow records to be linked according to the Container entity ID.
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Container Entity ID	1	Entity ID for container entity
7	Container Entity Instance	1	Instance ID for container entity
8	Container Entity Device Address	1	[7:1] - Slave address of management controller against which the device-relative Entity Instance for the container entity is defined. [0] - reserved, write as 0b Set to 00h if Instance ID is device-relative.
9	Container Entity Device Channel	1	[7:4] - Channel number of the channel that holds the management controller against which the device-relative Entity Instance for the container entity is defined. 0h if Instance ID is device-relative. [3:0] - reserved, write as 0000b.
10	flags	1	[7] - 0b = contained entities specified as list 1b = contained entities specified as range [6] - Record Link 0b = no linked Entity Association records 1b = linked Entity Association records exist [5] - 0b = Container entity and contained entities can be assumed absent if presence sensor for container entity cannot be accessed. This value is also used if the entity does not have a presence sensor. 1b = Presence sensor should always be accessible. Software should consider it an error if the presence sensor associated with the container entity is not accessible. If a presence sensor is accessible, then the presence sensor can still report that the container entity is absent. [4:0] - reserved, write as 00000b
11	Contained Entity 1 Device Address	1	[7:1] - Slave address of management controller against which the device-relative Entity Instance for contained entity 1 is defined. [0] - reserved, write as 0b Set to 00h if Entity Instance values are device-relative.
12	Contained Entity 1 Device Channel	1	[7:4] - Channel number of the channel that holds the management controller against which the device-relative Entity Instance for contained entity 1 is defined. 0h if Entity Instance values are device-relative. [3:0] - reserved, write as 0000b
13	Contained Entity 1 / Range 1 Entity ID	1	If list: Entity ID for contained entity 1 If range: Entity ID of entity for contained entity range 1
14	Contained Entity 1 Instance Range 1 first entity instance	1	If list: Entity Instance for contained entity 1 If range: Entity Instance for first entity in contained entity range 1

	RECORD BODY BYTES		
15	Contained Entity 2 Device Address	1	[7:1] - Slave address of management controller against which the device-relative Entity Instance for contained entity 2 is defined. [0] - reserved, write as 0b Set to 00h if Entity Instance values are device-relative or if the contained Entity entry is unspecified.
16	Contained Entity 2 Device Channel	1	[7:4] - Channel number of the channel that holds the management controller against which the device-relative Entity Instance for contained entity 2 is defined. 0h if Instance ID is device-relative, if the contained Entity entry is unspecified. [3:0] - reserved, write as 0000b
17	Contained Entity 2 / Range 1 Entity ID	1	00h = unspecified If list: Entity ID for contained entity 2 If range: Entity ID of entity for contained entity range 1 (must match byte 13)
18	Contained Entity 2 Instance / Range 1 last Entity Instance	1	00h = unspecified If list: Entity Instance for contained entity 2 If range: Entity Instance for last entity in contained entity range 1
19	Contained Entity 3 Device Address	1	[7:1] - Slave address of management controller against which the device-relative Entity Instance for contained entity 3 is defined. [0] - reserved, write as 0b Set to 00h if Entity Instance values are device-relative or if the contained Entity entry is unspecified.
20	Contained Entity 3 Device Channel	1	[7:4] - Channel number of the channel that holds the management controller against which the device-relative Entity Instance for contained entity 3 is defined. 0h if Instance ID is device-relative, if the contained Entity entry is unspecified. [3:0] - reserved, write as 0000b
21	Contained Entity 3 / Range 2 entity	1	00h = unspecified If list: Entity ID for contained entity 3 If range: Entity ID of entity for contained entity range 2
22	Contained Entity 3 Instance / Range 2 first entity Instance	1	00h = unspecified If list: Entity Instance for contained entity 3 If range: Entity Instance for first entity in contained entity range 2
23	Contained Entity 4 Device Address	1	[7:1] - Slave address of management controller against which the device-relative Entity Instance for contained entity 4 is defined. [0] - reserved, write as 0b Set to 00h if Entity Instance values are device-relative or if the contained Entity entry is unspecified.
24	Contained Entity 4 Device Channel	1	[7:4] - Channel number of the channel that holds the management controller against which the device-relative Entity Instance for contained entity 4 is defined. 0h if Instance ID is device-relative, if the contained Entity entry is unspecified. [3:0] - reserved, write as 0000b
25	Contained Entity 4 / Range 2 entity	1	00h = unspecified If list: Entity ID for contained entity 4 If range: Entity ID for entity for contained entity range 2 (must match byte 21)
26	Contained Entity 4 Instance / Range 2 last entity Instance	1	00h = unspecified If list: Entity Instance for contained entity 4 If range: Entity Instance for last entity in contained entity range 2
27:32	reserved	6	reserved

37.5 SDR Type 0Ah:0Fh - Reserved Records

This range and all other unspecified SDR Type values are reserved.

37.6 SDR Type 10h - Generic Device Locator Record

This record is used to store the location and type information for devices on the IPMB or management controller private busses that are neither IPMI FRU devices nor IPMI management controllers. These devices can either be common non-intelligent I^2C devices, special management ASICs, or proprietary controllers.

IPMI FRU Devices and Management Controllers are located via the *FRU Device Locator* and *Management Controller Device Locator* records described in following sections.

Table 37-5, Generic Device Locator Record - SDR Type 10h

byte	Field Name	size	Description
	RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. This may not actually be stored, but may be calculated when records are accessed.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *This is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits. E.g. 51h corresponds to "1.5".*
4	Record Type	1	Record Type Number = 10h, Device Locator
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Device Access Address	1	[7:1] - Slave address of management controller used to access device. 0000000b if device is directly on IPMB. [0] - reserved
7	Device Slave Address	1	[7:1] - 7-bit I^2C Slave Address[1]. This is relative to the bus the device is on. For devices on the IPMB, this is the slave address of the device on the IPMB. For devices on a private bus, this is the slave address of the device on the private bus. [0] - Channel Number ms-bit (For IPMI 1.5. This bit was reserved for IPMI v1.0.)
8	Access LUN / Bus ID	1	[7:5] - Channel Number ls-3 bits. Channel number for management controller used to access device. 0000b if device directly on the primary IPMB, or if controller is on the primary IPMB. (Note ms-bit of Channel Number is in Device Slave Address byte) [4:3] - LUN for Master Write-Read command. 00b if device is non-intelligent device directly on IPMB. [2:0] - Private bus ID if bus = Private. 000b if device directly on IPMB.
	RECORD BODY BYTES		
9	Address span	1	[7:3] - reserved [2:0] - number of additional consecutive slave addresses used by device. 00 indicates device only uses single address.
10	reserved	1	reserved
11	Device Type	1	See *Table 37-11, IPMB/I2C Device Type Codes*
12	Device Type Modifier	1	See *Table 37-11, IPMB/I2C Device Type Codes*
13	Entity ID	1	Entity ID for the FRU associated with this device. 00h if not specified.
14	Entity Instance	1	Instance number for entity.
15	OEM	1	Reserved for OEM use.
16	Device ID String Type/Length	1	Device ID String Type/Length code per Section *37.14, Type/Length Byte Format.*
17: +N	Device ID String	N	Short 'ID' string for the device. *16 bytes, maximum.*

Notes:

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

37.7 SDR Type 11h - FRU Device Locator Record

This record is used for locating FRU information that is on the IPMB, on a Private Bus behind or management controller, or accessed via FRU commands to a management controller. This excludes any FRU Information that is accessed via FRU commands at LUN 00b of a management controller. The presence or absence of that FRU Information is indicated using the Management Device Locator record (see *Table 37-7, Management Controller Device Locator - SDR 12h*, below).

Table 37-6, FRU Device Locator Record - SDR Type 11h

byte	Field Name	size	Description
	RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. This may not actually be stored, but may be calculated when records are accessed.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *This is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits. E.g. 51h corresponds to "1.5".*
4	Record Type	1	Record Type Number =11h, FRU Device Locator
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Device Access Address	1	[7:1] - Slave address of controller used to access device. 0000000b if device is directly on IPMB. (This field indicates whether the device is on a private bus or not) [0] - reserved
7	FRU Device ID / Device Slave Address	1	For LOGICAL FRU DEVICE (accessed via FRU commands to mgmt. controller): [7:0] - Number identifying FRU device within given IPM Controller. FFh = reserved. The primary FRU device for a management controller is always device #0 at LUN 00b. The primary FRU device is *not* reported via this FRU Device Locator record - its presence is identified via the Device Capabilities field in the Management Controller Device Locator record. For non-intelligent FRU device: [7:1] - 7-bit I^2C Slave Address[1] . This is relative to the bus the device is on. For devices on the IPMB, this is the slave address of the device on the IPMB. For devices on a private bus, this is the slave address of the device on the private bus. [0] - reserved
8	Logical-Physical / Access LUN / Bus ID	1	[7] - logical/physical FRU device 0b = device is not a logical FRU Device 1b = device is logical FRU Device (accessed via FRU commands to mgmt. controller) [6:5] - reserved. [4:3] - LUN for Master Write-Read command or FRU Command. 00b if device is non-intelligent device directly on IPMB. [2:0] - Private bus ID if bus = Private. 000b if device directly on IPMB, or device is a logical FRU Device.
9	Channel Number	1	[7:4] - Channel number for management controller used to access device. 000b if device directly on the primary IPMB, or if controller is on the primary IPMB. Ms-bit for channel number is kept in next byte. (For IPMI v1.5. This byte position was reserved for IPMI v1.0.) [3:0] - reserved
	RECORD BODY BYTES		
10	reserved	1	reserved
11	Device Type	1	See *Table 37-11, IPMB/I^2C Device Type Codes*. 10h for Logical FRU Device.
12	Device Type Modifier	1	See *Table 37-11, IPMB/I^2C Device Type Codes.*
13	FRU Entity ID	1	Entity ID for the device associated with this FRU information.
14	FRU Entity Instance	1	Instance number for entity.
15	OEM	1	Reserved for OEM use.

16	Device ID String Type/Length	1	Device ID String Type/Length code per *Section 37.14, Type/Length Byte Format*.
17: +N	Device String	N	Short 'ID' string for the FRU Device. *16 bytes, maximum.*

Notes:

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

37.8 SDR Type 12h - Management Controller Device Locator Record

This information is used for identifying management controllers on the IPMB and other internal channels, and for providing Entity and initialization information for all management controllers, including the BMC.

Table 37-7, Management Controller Device Locator - SDR 12h

byte	Field Name	size	Description
	RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. This may not actually be stored, but may be calculated when records are accessed.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *This is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits. E.g. 51h corresponds to "1.5".*
4	Record Type	1	Record Type Number = 12h, Management Controller Locator
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Device Slave Address	1	[7:1] - 7-bit I^2C Slave Address[1] of device on channel. [0] - reserved.
7	Channel Number	1	[7:4] - reserved [3:0] - Channel number for the channel that the management controller is on. Use 0h for the primary BMC. (New byte for IPMI v1.5. Note this addition causes some of the following byte offsets to be pushed down when compared to the IPMI v1.0 version of this record.)
	RECORD BODY BYTES		
8	Power State Notification Global Initialization	1	Power State Notification [7] - 1b = ACPI System Power State notification required (by system s/w) 0b = no ACPI System Power State notification required [6] - 1b = ACPI Device Power State notification required (by system s/w) 0b = no ACPI Device Power State notification required [5] - For backward compatibility, this bit does not apply to the BMC, and should be written as 0b. 0b = Dynamic controller - controller may or may not be present. Software should not generate error status if this controller is not present. 1b = Static controller - this controller is expected to be present in the system at all times. Software may generate an error status if controller is not detected. [4] - reserved Global Initialization [3] - 1b = Controller logs Initialization Agent errors (only applicable to Management Controller that implements the initialization agent function. Set to 0b otherwise.) [2] - 1b = Log Initialization Agent errors accessing this controller (this directs the initialization agent to log any failures setting the Event Receiver) [1:0] - 00b = Enable event message generation from controller (Init agent will set Event Receiver address into controller) 01b = Disable event message generation from controller (Init agent will set Event Receiver to FFh). This provides a temporary fix for a broken controller that floods the system with events. It can also be used for development / debug purposes. 10b = Do not initialize controller. This selection is for development / debug support. 11b = reserved.

byte	Field Name	size	Description
9	Device Capabilities	1	Device Support [7] - 1b = Chassis Device. (device functions as chassis device, per ICMB spec) [6] - 1b = Bridge (Controller responds to Bridge NetFn commands) [5] - 1b = IPMB Event Generator (device generates event messages on IPMB) [4] - 1b = IPMB Event Receiver (device accepts event messages from IPMB) [3] - 1b = FRU Inventory Device (accepts FRU commands to FRU Device #0 at LUN 00b) [2] - 1b = SEL Device (provides interface to SEL) [1] - 1b = SDR Repository Device (For BMC, indicates BMC provides interface to 1b = SDR Repository. For other controller, indicates controller accepts Device SDR commands) [0] - 1b = Sensor Device (device accepts sensor commands) See *Table 37-11, IPMB/I²C Device Type Codes*
10	reserved	1	reserved
11	reserved	1	reserved
12	reserved	1	reserved
13	Entity ID	1	Entity ID for the FRU associated with this device. 00h if not specified. If device supports FRU commands at LUN 00b, this Entity ID applies to both the IPM device and the FRU information accessed via LUN 00b.
14	Entity Instance	1	Instance number for entity.
15	OEM	1	Reserved for OEM use.
16	Device ID String Type/Length	1	Device ID String Type/Length code per *Section 37.14, Type/Length Byte Format*.
17:+N	Device ID String	N	Short 'ID' string for the device. *16 bytes, maximum.*

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

37.9 SDR Type 13h - Management Controller Confirmation Record

This record can be used by utility software to record that a given controller has been discovered in the system. Later, the record information can be used by software to confirm that the same controller is still present.

Table 37-8, Management Controller Confirmation Record - SDR Type 13h

byte	Field Name	size	Description
	RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. This may not actually be stored, but may be calculated when records are accessed.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *This is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits. E.g. I.e. 51h corresponds to "1.5".*
4	Record Type	1	Record Type Number = 13h, Management Controller Confirmation
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Device Slave Address	1	[7:1] - 7-bit I^2C Slave Address[1] of device on IPMB. [0] - reserved. Write as 0b
7	Device ID	1	Device ID from *Get Device ID* command. 00h = unspecified.
8	Channel Number / Device Revision	1	[7:4] - Channel Number for channel that management controller is located on. Use 0h for the primary BMC. (New for IPMI v1.5) [3:0] - Device Revision from *Get Device ID* command, binary encoded.
	RECORD BODY BYTES		
9	Firmware Revision 1	1	Firmware Revision 1 from *Get Device ID* command. [7] - reserved. Do not compare against same bits returned from *Get Device ID* command. [6:0] - Major Firmware Revision, binary encoded.
10	Firmware Revision 2	1	Firmware Revision 2 from *Get Device ID* command. Minor Firmware Revision. BCD encoded.
11	IPMI Version	1	IPMI Version from *Get Device ID* command. Holds IPMI Command Specification Version. BCD encoded. 00h = reserved. Bits 7:4 hold the Least Significant digit of the revision, while bits 3:0 hold the Most Significant bits. E.g. a value of 01h indicates revision 1.0
12:14	Manufacturer ID	3	Manufacturer ID from *Get Device ID* command, LS Byte first. Most significant four bits = reserved (0000b). xFFFFFh = ignore Manufacturer ID. (use for IPMI v0.9 controllers that don't provide a Manufacturer ID)
15:16	Product ID	2	Product ID from *Get Device ID* command, LS Byte first. 0000h = unspecified. FFFFh = ignore Product ID. (use FFFFh for IPMI v0.9 controllers that don't provide a Manufacturer ID)
17:32	Device GUID	16	Device GUID from *Get Device GUID* command. Set to all 0's if controller doesn't support *Get Device GUID* command.

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

37.10 SDR Type 14h - BMC Message Channel Info Record

This record describes the allocation and type for the BMC message channels. This record type has been deprecated for IPMI v1.5. IPMI v1.5 systems should use the *Get Channel Info* command instead.

Table 37-9, BMC Message Channel Info Record - SDR Type 14h

byte	Field Name	size	Description
	RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. This may not actually be stored, but may be calculated when records are accessed.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **01h** for this specification. *This is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.* Note this record keeps the IPMI v1.0 version number.
4	Record Type	1	Record Type Number = 14h, BMC Message Channel Info
5	Record Length	1	Number of remaining record bytes following.
	RECORD BODY BYTES		
6	Message Channel 0 Info	1	Channel 0, if present, is pre-defined to be the channel used for communication with the IPMB. Thus, the Message Channel 0 Info field either has bits 3:0 = 0h, indicating 'channel not present', or a constant '10100001' (A1h) if an IPMB is present. The following bit definitions apply to the Message Channel Info fields for all channels: [7] - 1b = Transmit supported 0b = receive message queue access only [6:4] - Message Receive LUN 000b-011b = LUN to receive messages from this channel. 111b = no LUN associated with receiving messages from this channel. all other = reserved [3:0] - Channel Protocol - this indicates the data format messages received from the channel in the Receive Message Queue and the format of data to be used for the *Send Message* command. 0h = Channel not present / not used. 1h = IPMB 2h = ICMB v1.0 3h = ICMB v0.9 4h = SMBus v1.0 Host (the controller must accept being addressed as a slave, and accept the SMBus *Modified Write Word* protocol. The interface may optionally accept a full SMBus Write Block. An SMBus channel can simultaneously support low-level I^2C devices, but not IPMI devices) 5h = System Format (Request messages of the format defined by the system interface: e.g. NetFn/LUN, Command, Data. Response messages as: NetFn/LUN, Command, Completion Code, Data - with the same messages size limitations as standard IPMI messages delivered over the system interface.) Ch-Fh = OEM Protocol 1 through 4, respectively all other = reserved
7	Channel 1 Info	1	Message Channel 1 Info
8	Channel 2 Info	1	Message Channel 2 Info
9	Channel 3 Info	1	Message Channel 3 Info
10	Channel 4 Info	1	Message Channel 4 Info
11	Channel 5 Info	1	Message Channel 5 Info
12	Channel 6 Info	1	Message Channel 6 Info
13	Channel 7 Info	1	Message Channel 7 Info

byte	Field Name	size	Description
14	Messaging Interrupt Type	1	00h-0Fh = IRQ 0 through 15, respectively 10h-13h = PCI A-D, respectively 14h = SMI 15h = SCI 20h-5Fh = system interrupt 0 through 63, respectively 60h = assigned by ACPI / Plug 'n Play BIOS FFh = no interrupt all other = reserved
15	Event Message Buffer Interrupt Type	1	see types defined for Messaging Interrupt Type byte 14 in this record.
16	reserved	1	reserved

37.11 SDR Type C0h - OEM Record

These record type numbers are reserved for OEM definition. OEM defined records are limited to a *maximum of 64 bytes*, including the header.

Note: OEM unique records should be avoided when possible. The amount of space available for these record types is implementation dependent and may be limited.

Table 37-10, OEM Record - SDR Type C0h

byte	Field Name	size	Description
	RECORD HEADER		
1 2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. This may not actually be stored, but may be calculated when records are accessed.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *This is BCD encoded with bits 7:4 holding the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.*
4	Record Type	1	Record Type Number = C0h, OEM SDR
5	Record Length	1	Number of remaining record bytes following.
6 7 8	Manufacturer ID	3	Manufacturer ID code. LS Byte first. Most significant 4 bits = reserved (0000b). 000000h = unspecified, 0FFFFFh =reserved. This value is binary encoded. E.g. the ID for Intel Corporation is 343 decimal, which is 157h, which would be stored in this record as 57h, 01h, 00h for bytes 6 through 8, respectively.
9: 2+N	OEM Data	N	OEM Data. N bytes.

37.12 Device Type Codes

These codes are used to identify different types of devices on an IPMB, PCI Management Bus, or Private Management Bus connection to an IPMI management controller.

Table 37-11, IPMB/I²C Device Type Codes

Code	IPMB Device Type	Device Type Modifier
00h	reserved.	n/a
01h	reserved.	n/a
02h	DS1624 temperature sensor / EEPROM or equivalent	00h = unspecified
03h	DS1621 temperature sensor or equivalent	00h = unspecified
04h	LM75 Temperature Sensor or equivalent	00h = unspecified
05h	'Heceta' ASIC or similar	00h = Heceta 1 e.g. LM78 01h = Heceta 2 e.g. LM79 02h = LM80 03h = Heceta 3 e.g. LM81/ ADM9240 / DS1780 04h = Heceta 4 05h = Heceta 5
06h-07h	reserved	n/a
08h	EEPROM, 24C01 or equivalent	EEPROM Use: 00h = unspecified 01h = DIMM Memory ID 02h = IPMI FRU Inventory 03h = System Processor Cartridge FRU / PIROM (processor information ROM) all other = reserved
09h	EEPROM, 24C02 or equivalent	same as for code 08h
0Ah	EEPROM, 24C04 or equivalent	same as for code 08h
0Bh	EEPROM, 24C08 or equivalent	same as for code 08h
0Ch	EEPROM, 24C16 or equivalent	same as for code 08h
0Dh	EEPROM, 24C17 or equivalent	same as for code 08h
0Eh	EEPROM, 24C32 or equivalent	same as for code 08h
0Fh	EEPROM, 24C64 or equivalent	same as for code 08h
10h	FRU Inventory Device behind management controller (accessed using Read/Write FRU commands at LUN other than 00b)	00h = IPMI FRU Inventory [1] 01h = DIMM Memory ID 02h = IPMI FRU Inventory[1] 03h = System Processor Cartridge FRU / PIROM (processor information ROM) all other = reserved FFh = unspecified
11h-13h	reserved	n/a
14h	PCF 8570 256 byte RAM or equivalent	00h = unspecified
15h	PCF 8573 clock/calendar or equivalent	00h = unspecified
16h	PCF 8574A 'i/o port' or equivalent	00h = unspecified
17h	PCF 8583 clock/calendar or equivalent	00h = unspecified
18h	PCF 8593 clock/calendar or equivalent	00h = unspecified
19h	Clock calendar, type not specified	00h = unspecified
1Ah	PCF 8591 A/D, D/A Converter or equivalent	00h = unspecified
1Bh	i/o port, specific device not specified	00h = unspecified
1Ch	A/D Converter, specific device not specified	00h = unspecified
1Dh	D/A Converter, specific device not specified	00h = unspecified
1Eh	A/D, D/A Converter, specific device not specified	00h = unspecified
1Fh	LCD controller / Driver, specific device not specified	00h = unspecified
20h	Core Logic (Chip set) Device, specific device not specified	00h = unspecified
21h	LMC6874 Intelligent Battery controller, or equivalent	00h = unspecified
22h	Intelligent Battery controller, specific device not specified	00h = unspecified
23h	Combo Management ASIC, specific device not specified	00h = unspecified
24h	Maxim 1617 Temperature Sensor	00h = unspecified
BFh	Other / unspecified device	00h = unspecified
C0h - FFh	OEM specified device	OEM specific

all other	reserved	n/a

1. Either value can be used. The 00h Device Type Modifier is present for backward compatibility. The remaining modifiers line up with those for the 08h-0Fh Device Types.

37.13 Entity IDs

The Entity ID field is used for identifying the physical entity that a sensor or device is associated with. If multiple sensors refer to the same entity, they will have the same Entity ID field value. For example, if a voltage sensor and a temperature sensor are both for a 'Power Supply 1' entity the Entity ID in their sensor data records would both be 10 (0Ah), per the Entity ID table.

Table 37-12, Entity ID Codes

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

Code		Entity
38	26h	Remote (Out of Band) Management Communication Device
39	27h	External Environment - This Entity ID can be used to identify the environment outside the system chassis. For example, a system may have a temperature sensor that monitors the temperature "outside the box". Such a temperature sensor can be associated with an External Environment entity. This value will typically be used as a single instance physical entity. However, the Entity Instance value can be used to denote a difference in regions of the external environment. For example, the region around the front of a chassis may be considered to be different from the region around the back, in which case it would be reasonable to have two different instances of the External Environment entity.
40	28h	battery
	90h-AFh	Chassis-specific Entities. These IDs are system specific and can be assigned by the chassis provider.
	B0h-CFh	Board-set specific Entities. These IDs are system specific and can be assigned by the Board-set provider.
	D0h-FFh	OEM System Integrator defined. These IDs are system specific and can be assigned by the system integrator, or OEM.
-		all other values reserved

* = DMI standard groups compatible. These codes can be mapped to corresponding codes in the DMI *Systems Standard Groups Definition* MIF.

37.14 Type/Length Byte Format

The type/length byte is a variation of the type/length byte format defined in the *Platform Management FRU Information Storage Definition.* The main differences being that bit 5 is reserved in the IPMI specification type/length byte, where it is part of the length field in the *Platform Management FRU* specification, and bits 7:6 = 00b define a Unicode string in the IPMI specification, whereas they specify a binary field in the *Platform Management FRU* specification.

Type/Length Byte definition:

7:6 00 = Unicode
01 = BCD plus (see below)
10 = 6-bit ASCII, packed
11 = 8-bit ASCII + Latin 1. At least two bytes of data must be present when this type is used. Therefore, the length (number of data bytes) will be >1 if data is present, 0 if data is not present. A length of 1 is reserved.

5 reserved.

4:0 length of following data, in *characters*. 00000b indicates 'none following'. 11111b = reserved.

BCD PLUS definition:

0h - 9h = digits 0 through 9
Ah = space
Bh = dash '-'
Ch = period '.'
Dh = colon ':'
Eh = comma ','
Fh = underscore '_'

Table 37-13, 6-bit ASCII definition

0	sp	*10*	0	*20*	@	*30*	P
1	!	*11*	1	*21*	A	*31*	Q
2	"	*12*	2	*22*	B	*32*	R
3	#	*13*	3	*23*	C	*33*	S
4	$	*14*	4	*24*	D	*34*	T
5	%	*15*	5	*25*	E	*35*	U
6	&	*16*	6	*26*	F	*36*	V
7	'	*17*	7	*27*	G	*37*	W
8	(	*18*	8	*28*	H	*38*	X
9	)	*19*	9	*29*	I	*39*	Y
A	*	*1A*	:	*2A*	J	*3A*	Z
B	+	*1B*	;	*2B*	K	*3B*	[
C	,	*1C*	<	*2C*	L	*3C*	\

D	-	*1D*	=	*2D*	M	*3D*	]
E	.	*1E*	>	*2E*	N	*3E*	^
F	/	*1F*	?	*2F*	O	*3F*	_

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

"6-bit ASCII" is the 64 characters starting from character 20h (space) from the ASCII+LATIN1 set. So 6-bit ASCII value 000000b maps to 20h (space), 000001b maps to 21h (!), etc. Packed 6-bit ASCII takes the 6-bit characters and packs them 4 characters to every 3 bytes, with the first character in the least significant 6-bits of the first byte. A table of 6-bit ASCII codes and an example of packed 6-bit ASCII characters follows:

37.15 6-bit ASCII Packing Example

"IPMI" encoded in 6-bit ASCII is:

I = 29h (101001b)

P = 30h (110000b)

M = 2Dh (101101b)

I = 29h (101001b)

Which gets packed into bytes as follows:

Figure 37-1, 6-bit Packed ASCII Example

bit	7	6	5	4	3	2	1	0	hex
byte 1	0	0	***1***	***0***	***1***	***0***	***0***	***1***	29h
byte 2	**1**	**1**	**0**	**1**	1	1	0	0	DCh
byte 3	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	A6h

37.16 Sensor Unit Type Codes

The following type codes are encodes units from the International System of Units, selected additional 'imperial' measures, and common 'computer' and communication measurements.

Table 37-14, Sensor Unit Type Codes

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

38. Examples

38.1 Processor Sensor with Sensor-specific States & Event Generation

The following example shows a Processor sensor that has *sensor specific* discrete state information. The Sensor Type Code for Processor is 07h. A sensor that uses sensor-specific state information is identified using an Event/Reading Type Code of **6Fh** in the SDR.

Sensor Type:	Processor	= 07h (From *Table 36-3, Sensor Type Codes*)
Event/Reading Type:	Sensor-specific	= 6Fh (From *Table 36-1, Event/Reading Type Code Ranges*)

The example processor sensor returns the following readings

IERR, Thermal Trip, and Processor Presence

and generates the following events:

IERR Asserted, Processor Presence Asserted, Processor Presence Deasserted

The Reading Mask and Event Mask fields in the SDR for the sensor is used to tell System Management Software that these are the only possible readings that the sensor will return. System Management Software can use this information to customize the way it displays or acts on the sensor state and sensor events. The same State Bit field positions that are used for the masks are also used in commands for accessing and configuring the sensor, such as the *Get Sensor Reading* and *Set Sensor Event Enable* commands.

Note that, for this example, the events that can be generated are a subset of the possible states that can be read from the sensor, and that the deassertion events don't necessarily match up with the assertion events. Most sensors will be this way. In fact, in a typical implementation most sensors will not generate any deassertion events. The guideline is that warning and error conditions should generate Event Messages (and be logged) while non-critical or informational state changes should not. This helps ensure that the event log and event receiver does not get clogged up with non-critical information. Event Messages do not utilize the state bit field directly - but instead use an 4-bit *offset* value corresponding to the State Bit position of the state change that triggered the event. The 4-bit offset helps keep Event Messages compact, allowing for additional parameter bytes to be passed with the event while keeping the size of SEL Event Records at 16 bytes. While a discrete sensor can simultaneously track and report multiple states, a consequence of using an offset in the Event Message is that only one state change event at a time gets reported in an Event Message.

An *Event Dir* bit used in Event Messages and SEL Event Records indicates whether the event was an assertion event (0) or a deassertion event (1). For example, if a Processor Presence Detection event occurred the Event Message would contain an offset value of 7, with an event dir bit of 0.

Table 38-1, Example discrete Processor sensor with Sensor-specific states & event generation

State	Offset	State Bit	SDR Reading Mask	SDR Assertion Event Mask	SDR Deassertion Event Mask
IERR	0h	0	**1**	**1**	0
Thermal Trip	1h	1	**1**	0	0
FRB1/BIST failure	2h	2	0	0	0
FRB2/Hang in POST failure	3h	3	0	0	0
FRB3/Processor Startup/Init failure	4h	4	0	0	0
Configuration Error (for DMI)	5h	5	0	0	0
SM BIOS 'Uncorrectable CPU-complex Error'	6h	6	0	0	0
Processor Presence detected	7h	7	**1**	**1**	**1**
Processor disabled	8h	8	0	0	0
Terminator Presence Detected	9h	9	0	0	0
unspecified	Ah	10	0	0	0
unspecified	Bh	11	0	0	0
unspecified	Ch	12	0	0	0
unspecified	Dh	13	0	0	0
unspecified	Eh	14	0	0	0

38.2 Processor Sensor with Generic States & Event Generation

Even though the Processor sensor type has sensor-specific states defined, that doesn't mean you have to use them. You can use the generic discrete states with any sensor type. In this example, we show the definition of a Processor sensor that returns a generic severity state.

Sensor Type:	Processor	07h (from *Table 36-3, Sensor Type Codes*)
Event/Reading Type:	generic Severity	07h (from *Table 36-1, Event/Reading Type Code Ranges* and *Table 36-2, Generic Event/Reading Type Codes*)

In this example, assume that we return the following Severity status:

OK, Non-Critical from OK, Non-Critical from more severe, Critical from less severe

and generate events on the following transitions

Non-Critical from OK, Critical from less severe

Note that this definition only generates events on worsening severity conditions. This is a recommended practice to avoid filling the SEL with non-failure related information.

The following table shows the event offsets returned in event messages, the state bits returned from the 'Get Sensor Reading' command, and the Assertion / Deassertion Masks in the SDR corresponding to a sensor with the example characteristics.

Table 38-2, Example discrete Processor sensor with Generic states & event generation

State	Offset	State Bit	SDR Reading Mask	SDR Assertion Event Mask	SDR Deassertion Event Mask
transition to OK	0h	0	**1**	0	0
transition to Non-Critical from OK	1h	1	**1**	**1**	0
transition to Critical from less severe	2h	2	**1**	**1**	0
transition to Non-recoverable from less severe	3h	3	0	0	0
transition to Non-Critical from more severe	4h	4	**1**	0	0
transition to Critical from Non-recoverable	5h	5	0	0	0
transition to Non-recoverable	6h	6	0	0	0
Monitor	7h	7	0	0	0
Informational	8h	8	0	0	0
reserved	9h	9	0	0	0
reserved	Ah	10	0	0	0
reserved	Bh	11	0	0	0
reserved	Ch	12	0	0	0
reserved	Dh	13	0	0	0
reserved	Eh	14	0	0	0

Appendix A - Previous Sequence Number Tracking

The following illustrates how a method for tracking the last eight received sequence numbers can be used for handling out-of-sequence packet reception. The method illustration assumes that the receiver tracks the highest received sequence number that has been accepted, shown as 'Highest Received' column, and which of the previous eight sequence numbers have been received, shown as the 'Previously Received List'.

Note that for implementation, the previously received list can be implemented as a bit field where 1=received, 0=not received, and the bit positions correspond to sequence numbers for Highest Received-1, Highest Received-2, Highest Received-3, etc. The handling of wrap-around of the sequence number is not shown and is left to the reader.

1. Startup initialization. Highest Received session sequence number is set to value that was sent with the *Activate Session* command. The Previously Received List is initialized as if the preceding eight sequence numbers were received. This prevents a packet with a sequence number less than the initial value from being accepted. For this example, assume the initial sequence number is 40.

   Initial values are:

Highest Received	Previously Received List							
40	39	38	37	36	35	34	33	32
	Y	Y	Y	Y	Y	Y	Y	Y

2. Packet 41 is received. Packet is accepted because it is no more than 8 counts greater than the Highest Received value from step 1. Highest Received becomes 41. Previously received list gets 'pushed up'. Updated values are:

Highest Received	Previously Received List							
41	40	39	38	37	36	35	34	33
	Y	Y	Y	Y	Y	Y	Y	Y

3. Packet 44 received. Accepted because it is within 8 counts of the last Highest Received. Highest Received becomes 44. Previous received list gets pushed up by 2. Note that packets 43 and 42 are listed as unreceived. Updated values are:

Highest Received	Previously Received List							
44	43	42	41	40	39	38	37	36
	N	N	Y	Y	Y	Y	Y	Y

4. Packet 39 received. Dropped because it is a duplicate with previously received packet. No update to Highest Received and Previously Received List. Updated values are:

Highest Received	Previously Received List							
44	43	42	41	40	39	38	37	36
	N	N	Y	Y	Y	Y	Y	Y

5. Packet 42 received. Accepted because it is listed as unreceived on the Previously Received List. Packet 42 listed as received. No update to Highest Received because 42 is not higher than 44. Updated values are:

Highest Received	Previously Received List							
44	43	42	41	40	39	38	37	36
	N	Y	Y	Y	Y	Y	Y	Y

Appendix B - Example PEF Mask Compare Algorithm

This is an 'untested' algorithm provided as a starting point for guiding an implementation. While this looks a lot like C, consider this as pseudo-code - or just read the comments...

Figure B-1, Example Event Data Comparison Algorithm

```
//-------- here we go -----------------------------------------
// First, make a value that has the 'don't care' bits forced to 0
temp1 = (test_value & AND_mask);

// Next, check it for a match with the 'exact match' bits
// (AND'ing with compare1 forces the 'non-exact' bit positions to 0.
// By AND'ing temp1 with compare 1, and compare2 with compare1 the result is two
// values that both have the 'non-exact' bit positions forced to 0. The
// remaining non-forced bit positions should match.
if ((temp1 & compare1)==(compare2 & compare1)) { // they match
 match = true;    // so far! this may change! (innocent until proven guilty)

 // Now see if there are 'non exact' bits to check
 if (compare1 != 0xFF) {    // yep, there are non-exact bits to check

    // Make a value that has both the 'don't care' and the 'exact compare'
      // bits forced to 0.
  temp2 = temp1 & !compare1;

// then AND it with compare2. If the result is non-zero, you
// had at least one '1' in the right place.
// (But first check if compare2 is 0. If so, there are no 1's to check for.)
  if (compare2 != 0x00) {
    if !(temp2 & compare2) match = false; // No 1's in right places
  };

// Take temp2 and AND with NOT compare2 to look for 0's.
// (But first check if compare2 is FF. If so, no 0's to check for.)
if (compare2 != 0xFF) {
    if !(temp2 & !compare2) match = false; // No 0's in right places
  };
 };
} else (match = false); // 'exact match' bits didn't match

//------------------------------------------------------------------------------
// somewhat nastier condensed version (test_value variable re-use, logical test
// used for detecting non-zero values, etc.)
// sets variable "match":  0 = no match, non-zero = match.
//------------------------------------------------------------------------------
testValue &= AND_mask;            // force don't care bits to 0
// then check the 'exact match' bits
if ((testValue & compare1)==(compare2 & compare1)){
 match = 1;                 // so far, so good
 if (compare1 != 0xFF) {          // we have 'non-exact' bits to check
  testValue &= !compare1;       // force 'exact match' bits to 0, too
 if (compare2) match = testValue & compare2; // look for 1's in right places
 // look for 0's in right places, but not unless we still have a match
  if (match && (compare2 != 0xFF)) match = !(testValue & !compare2);
 };
} else (match=0);
```

Appendix C - Locating IPMI System Interfaces via SM BIOS Tables

The *System Management BIOS Reference Specification*, Version 2.3.1, March 16, 1999 (hereon referred to as SM BIOS) includes the following optional record for identifying the initial location of IPMI system interfaces and interrupts. This is summarized in the following table. Fields in BOLD represent fields that are additions to the 2.3.1 specification. See [SMBIOS] for other application information on SM BIOS.

Note that the settings that this structure reports may be over-ridden by 'Plug-and-Play' reassignment by the OS. Therefore, this structure should be used only when the interface cannot be discovered via 'Plug-and-Play' discovery mechanisms incorporated in interfaces such as PCI and ACPI.
IPMI Device Information (Type 38).

Table C-1, SM BIOS IPMI Device Information Record

Offset	Name	Length	Value	Description
00h	Type	BYTE	38	IPMI Device Information structure indicator. (Note this number is given in decimal)
01h	Length	BYTE		Length of the structure, a minimum of 10h (for full IPMI address description, this is a minimum of 12h)
02h	Handle	WORD	Varies	
04h	Interface Type	BYTE	ENUM	Baseboard Management Controller (BMC) interface type, see *Table C-2, Interface Type field values*, below.
05h	IPMI Specification Revision	BYTE	Varies	Somewhat mis-named. Actually identifies the IPMI Specification *Version*, in BCD format, to which the BMC was designed. Bits 7:4 hold the most significant digit of the version, while bits 3:0 hold the least significant bits, e.g. a value of 15h indicates version 1.5.
06h	I^2C Slave Address	BYTE	Varies	The slave address on the I^2C bus of this BMC.
07h	NV Storage Device Address	BYTE	Varies	Bus id of the NV storage device. If no storage device exists for this BMC, the field is set to 0FFh.
08h	Base Address	QWORD	Varies	Identifies the base address (either memory-mapped or I/O) of the BMC. If the least-significant bit of the field is a 1, the address is in I/O space; otherwise, the address is memory-mapped. Refer to the *IPMI Interface Specification* for usage details.
10h	**Base Address Modifier / Interrupt Info**	**BYTE**	**Varies**	**Base Address Modifier** **bit 7:6 - Register spacing** **00b = interface registers are on successive byte boundaries** **01b = interface registers are on 32-bit boundaries** **10b = interface registers are on 16-byte boundaries** **11b = reserved** **bit 5 - reserved. Return as 0b.** **bit 4 - LS-bit for addresses** **0b = Address bit 0 = 0b** **1b = Address bit 0 = 1b**

				Interrupt Info **Identifies the type and polarity of the interrupt associated with the IPMI system interface, if any.** **bit 3 - 1b = interrupt info specified** **0b = interrupt info not specified** **bit 2 - reserved. Return as 0b.** **bit 1 - Interrupt Polarity.** **1b = active high, 0b = active low.** **bit 0 - Interrupt Trigger Mode.** **1b = level, 0b = edge.**
11h	**Interrupt Number**	**BYTE**	**Varies**	**Interrupt number for IPMI System Interface.** **00h = unspecified / unsupported**

C.1 IPMI Device Information - BMC Interface

The following sections present more information describing the Type 38 record fields and their use.

C.1.1 Interface Type

The following table presents the meaning of the values for the Interface Type Field:

Table C-2, Interface Type field values

Byte Value	Meaning
00h	Unknown
01h	KCS: Keyboard Controller Style
02h	SMIC: Server Management Interface Chip
03h	BT: Block Transfer
04h to 0FFh	Reserved for future assignment by this specification

C.1.2 IPMI Specification Revision Field

Identifies the IPMI Specification Revision, in BCD format, to which the BMC was designed. Bits 7:4 hold the most significant digit of the revision, while bits 3:0 hold the least significant bits, e.g. a value of 10h indicates revision 1.0.

C.1.3 I^2C Slave Address Field

This field indicates the slave address of the BMC on the primary IPMB in the system. The most significant seven bits hold the address. The least significant bit is reserved and shall be returned as 0b. The 7-bit portion of the slave address for the BMC is 0010 000_b, therefore this field will typically be populated with the value 20h.

C.1.4 NV Storage Device Address Field

The field is reserved for use by the System Integrator (party that integrates motherboard and chassis).
This field describes the location of an auxiliary OEM NV Storage Device on the primary IPMB in the system.

C.1.5 Base Address Field

This field is used to describe the base address for the BMC's system interface. The field can describe both I/O mapped and memory-mapped base addresses. The least significant bit of this field indicates whether the base address is an I/O address or a memory address. The most significant 63-bits of this field holds the most significant 63 bits (bits 63:1) of a 64-bit address. The least significant bit (bit 0) of the base address is kept in the Base Address Modifier field.

All IPMI system interface registers are inherently non-cacheable and the register locations must be implemented as non-cacheable addresses.

C.1.6 Base Address Modifier Field

This field provides the least-significant bit for the base address, information indicating how the system interface registers are aligned (either on byte, 32-bit, or 16-byte boundaries).

C.1.7 System Interface Register Alignment

System interface registers can optionally be defined on 32-bit or 16-byte boundaries. In this case, the registers are 32-bits (4 bytes) apart. Base Addresses must match the specified register alignment. For example, the base address for a 32-bit aligned interface must have its two least significant address bits = 00b. Thus, the LS bit field in the Base Address Modifier is always 0b for non-byte-aligned addresses.

C.1.7.1 Byte-spaced I/O Address Examples

The following example shows how the default system interface addresses would be represented in the SM BIOS Base Address and Base Address Modified fields. Base Address bit 0 = 1b indicates that the base address is an I/O address. The default system interface definition specifies that the system interface registers occupy consecutive byte locations. Thus, the register spacing in the Base Address Modifier is set to 0b. Note that the LS bit field in the Base Address Modifier field matches the least-significant bit listed in the corresponding addresses from the Default Base Address column.

Table C-3, Byte-aligned I/O Mapped Register Address examples

Interface	Default Base Address	SM BIOS Base Address	LS bit field	Register spacing
KCS	0CA2h	0000 0000 0000 CA3h	0b	00b
SMIC	0CA9h	0000 0000 0000 CA9h	1b	00b
Block Transfer (BT)	00E4h	0000 0000 0000 00E5h	0b	00b

C.1.7.2 32-bit Spaced I/O Address Examples

The following example shows examples addresses for a KCS interface implemented with 32-bit aligned registers at I/O base address CACh.

Table C-4, 32-bit aligned I/O Mapped Register Address examples

	Example I/O Address	SM BIOS Base Address	LS bit field	Register spacing
base address	0000 0CACh	0000 0000 0000 0CADh	0b	01b
Data_In	0000 0CACh	0000 0000 0000 0CADh	0b	01b
Data_Out	0000 0CACh	0000 0000 0000 0CADh	0b	01b
Command	0000 0CB0h	0000 0000 0000 0CB1h	0b	01b
Status	0000 0CB0h	0000 0000 0000 0CB1h	0b	01b

C.1.7.3 Memory-mapped Base Address

For memory-mapped system interfaces, the Base Address field and Base Address Modifier are used in the same manner as for an I/O-mapped interface, except that Base Address bit 0 is set to 0b.

C.1.7.4 Interrupt Info Field

This field identifies the type and polarity of the interrupt associated with the IPMI system interface, if any. Refer to the Type 38 table, above, for individual bit descriptions.

C.1.8 Interrupt Number Field

This field holds the interrupt number for the IPMI System Interface. The field is set to 00h when the number is unspecified or an interrupt is not supported.

Appendix D - Determining Message Size Requirements

Two factors drive the message size support requirements. The first is the message size limit of the IPMB. The IPMB is specified to have a 32-byte maximum overall message length (from slave address through the last checksum byte). The second is the ability for the *Master Write-Read* command to be able to be used to support SMBus 2.0 protocols for accessing slave devices and supporting SMBus 2.0 channels in the BMC.

The largest SMBus 2.0 transaction is the Block-Write with PEC (Packet Error Code) protocol transaction. The Block-Write with PEC protocol transaction requires 36 bytes (including slave address) to be transferred as a single write transaction on SMBus.

- IPMB Message : 32 bytes, maximum, including slave address.
- SMBus 2.0 Message: 36 bytes, maximum, including slave address.

Local system software can use the BMC as a low-level controller to access private management busses, the IPMB, and SMBus connections by sending a *Master Write-Read* command to the BMC through the system interface. Using a *Master Write-Read* command to deliver a full-size SMBus 2.0 Block-Write protocol transaction requires accepting a 40 byte message over the KCS system interface (see following figure). Note that while the SMBus message is 36 bytes overall, the Slave Address is already part of the *Master Write-Read* command, so only 35 bytes is shown in the write data portion of the message.

Figure D-1, SMBus Write-Block by Master Write-Read through KCS/SMIC

NetFn/LUN	Command	Bus ID	Slave Address	Read Count	[Write Data]	**total**
1	1	1	1 (Slave address for SMBus Write-Block with PEC)	1	35 (Command, byte count, data, & PEC for SMBus Write-Block with PEC)	**36**

Similarly, the BMC needs to accept 36 bytes on any connection where the BMC could be the target of an SMBus 2.0 Write-Block protocol.

The SMBus 2.0 Block-Read operation only requires 34 bytes of input. (Byte Count, 32 data bytes, and PEC). So a private management bus that accessed SMBus 2.0 devices as a slave would only need to support 34 input bytes. (Note there's no slave address read from a Read-Block because the device is acting as a slave on the bus.) Therefore, the following show the size of *Master Write-Read Response* required to be delivered from the IPMB and KCS interfaces:

Figure D-2, Master Write-Read Response via KCS/SMIC

NetFn/LUN	Command	Completion Code	Read Data	**total**
1	1	1	34 (bytecount, data, and PEC for SMBus Read-Block with PEC)	**37**

For comparison, the following shows the *Get Message Response* that would return an entire 32-byte message in IPMB format.

Figure D-3, Get Message Response via KCS/SMIC

NetFn/LUN	Command	Completion Code	Channel #	Read Data	**total**
1	1	1	1	32	**36**

LAN or PPP IPMI request message for a *Master Write-Read* message to perform an SMBus 2.0 Block-Write with PEC protocol transaction:

Figure D-4, Master Write-Read Request via LAN/PPP

Software ID	NetFn/ LUN	Check 1	RqSA	RqSeq /LUN	CMD	Bus ID	Slave Address	Read Count	[Write Data]	Check 2	**total**
1	1	1	1	1	1	1	1	1	35	1	**45**

LAN or PPP IPMI response message for a *Master Write-Read* response that returns data for an SMBus 2.0 Block-Read with PEC protocol transaction. The SMBus Block-Read with PEC protocol requires reading a maximum of 34 bytes from the bus (byte count, 32-bytes of data, and PEC).

Figure D-5 Master Write-Read Response via LAN/PPP

Software ID	NetFn/ LUN	Check 1	RqSA	RqSeq /LUN	CMD	completion code	[Read Data]	Check 2	**total**
1	1	1	1	1	1	1	34	1	**42**

Private Bus Write from IPMB. Maximum Write Data that is supported to a private management bus using the *Master Write-Read* command delivered via IPMB:

Figure D-6, Master Write-Read Response via LAN/PPP

RsSA	NetFn/ LUN	Check 1	RqSA	RqSeq /LUN	CMD	Bus ID	**Slave Address**	Read Count	**[Write Data], max**	Check 2
1	1	1	1	1	1	1	**1**	1	**22**	1

Appendix E - Terminal Mode Grammar

E.1 Notation

[x] = x occurs one or more times. E.g. x, xx, xxxxx
[x | y | z] = A set consisting of one or more occurrences of x, y, or z in any order or combination. E.g. z, zyz, xzyxyzzxy
(x | y) = Exclusive OR. Only one of x or y, but not both.
{ x } = Optionally present. x may or may not occur.
BOLD = A string literal. *Case insensitive* unless otherwise noted.

E.2 Grammar for Terminal Mode Input

input_line ::=	start cmd_prefix space (command | message) {space} input_termination_seq
start ::=	L_bracket
cmd-prefix ::=	**SYS**
command ::=	(password_cmd | set_cmd | reset_cmd | power_cmd | health_cmd | comset_cmd | oem_cmd)
message ::=	message_segment { { space } line_continue message_segment }
input_termination_seq ::=	stop input_newline
password_cmd ::=	login_cmd | null_login_cmd | logout_cmd
login_cmd ::=	**PWD** space **-U** space username {space password}
null_login_cmd ::=	**PWD** space **-N** {space password}
logout_cmd ::=	**PWD** space **-X**
set_cmd ::=	**SET** space (set_boot | set_tcfg)
set_boot ::=	**BOOT** space hex_pair
set_tcfg ::=	**TCFG** space { (set_volatile) | (set_non-volatile) }
set_volatile ::=	**-V** space hex_pair hex_pair
set_non-volatile ::=	**-N** space hex_pair hex_pair
reset_cmd ::=	**RESET** space
power_cmd ::=	**POWER** space (**ON** | **OFF**)
health_cmd ::=	**HEALTH** space **QUERY** {space opt_verbose}

oem_cmd ::= oem_id space printable

username ::= [alphanumeric | punctuation]
// username is limited to 16 characters. Spaces are not allowed in usernames that work with terminal mode.

password ::= [alphanumeric | punctuation]
// password is limited to 16 characters. Spaces are not allowed in the terminal mode password.

message_segment ::= hex-pair {space message_segment}

line_continue ::= backslash input_newline

stop ::= R_bracket

input_newline ::= (cr | null)
// note: A configuration option affects which of these in-termination options may be enabled at a time.

oem_id ::= hex_pair hex_pair hex_pair hex_pair hex_pair hex_pair

opt_verbose ::= **-V**

printable ::= [alphanumeric | punctuation | space]

alphanumeric ::= [digit | alpha]

punctuation ::= [**!** | double-quote | **#** | **$** | **%** | **&** | apostrophe | **(** | **)** | ***** | **+** | grave_accent | hyphen | period | / | **:** | **;** | < | > | **?** | **@** | backslash | ^ | underscore | **{** | vertical_bar | tilde]
// note that the L_bracket and R_bracket characters not part of this set.
-

alpha ::= **a**-**z** | **A**-**Z**

digit ::= **0**-**9**

hex_pair ::= hex hex

hex ::= [digit | **a**-**f** | **A**-**F**]

L_bracket ::= **[**

R_bracket ::= **]**

space ::= ‘ ’ //20h

E.3 Grammar for Terminal Mode Output

output_line ::= start (ok_response | error_response | OEM_response | handshake) output_termination_seq

output_termination_seq ::= stop out_termination

output_newline ::= (cr lf | cr | lf | null) // Only one type used at a time. Configurable.

ok_response ::= **OK** { {space} command_response }

error_response ::= **ERR** {space err_code}

command_response ::= {output_newline} printable {output_newline {printable}}

OEM_response ::= **SYS** space oem_id printable {output_newline {printable}}

handshake ::= **SYS**

err_code ::= hex-pair

oem_id ::= hex_pair hex_pair hex_pair hex_pair hex_pair hex_pair

printable ::= [alphanumeric | punctuation | space]

alphanumeric ::= [digit | alpha]

punctuation ::= [**!** | double-quote | **#** | **$** | **%** | **&** | apostrophe | **(** | **)** | ***** | **+** | grave_accent | hyphen | period | / | **:** | **;** | < | > | **?** | **@** | backslash | ^ | underscore | **{** | vertical_bar | tilde]
// note that L_bracket and R_bracket characters not part of this set.
-

alpha ::= [**a**-**z** | **A**-**Z**]

digit ::= [**0**-**9**]

hex_pair ::= hex hex

hex ::= [digit | **a**-**f** | **A**-**F**]

L_bracket ::= **[**

R_bracket ::= **]**

space ::= ‘ ’ //20h

Appendix F - TAP Flow Summary

The following table presents implementation notes and an overview of the flow of a TAP page from a BMC. The table is intended as a guideline and does not supercede the TAP specification. Refer to [TAP] for complete information on implementing the Telocator Access Protocol.

Table F-1, TAP Flow Summary

step	Remote Entry Device (BMC)	Paging Terminal (Paging Service)	Description
1	Dial up the paging service. Use the ATDT modem command to start the modem. The modem returns one of the following: CONNECT NO CARRIER OK		
2		Connection established	
3	<CR>		The carriage return <CR> is repeated at t1 (2) second intervals until the paging company responds with ID= at the correct baud rate or until n1 (3) transmissions of <CR> have been completed.
4		ID=	Request for ID should be returned within t2 (1) second of the receipt of <CR>. The paging service may send <CR>, <LF>, <CR><LF>, or <LF><CR> after the ID=. The BMC should ignore extra characters after the ID=. Refer to the TAP spec for implementation advice on handling <LF> characters that may be received from the paging service. The paging service waits up to t5 (8) seconds for response to "ID=". The paging service may resend "ID=" up to n3 (3) times if a proper response is not received.
5	ESC>PG1<CR> or for password entry: <ESC>PG1pppppp<CR>		<ESC> signifies that the BMC wants to communicate with the paging company in automatic mode. "PG" signifies the type of service to be accessed and the types of fields in the message. P indicates that the message contains a “Pager ID” field and G indicates presence of a message text field. The paging service provider determines whether a Pager ID is used with the message text. Note that PG will typically be used even if the Pager ID field is empty. The next character represents the type of terminal or device attempting to send the pages:

			'1' is a category using the same protocol. Refer to [TAP] for other field values. The 6-character pppppp stands for an up to six character alphanumeric password. Password is optional. The paging service provider determines whether a password should be used. When an incorrect logon sequence beginning with <ESC> is received, the paging company sends "ID=" back to indicate a retry is requested.
6		<Message Sequence> followed by either:	A message sequence is defines as a series of short messages separated by <CR>'s. The first characters of each response message are a three ASCII digit response code. A <CR> always follows the message sequence.
		<CR><ACK><CR>	Logon accepted
		or: <CR><NAK><CR>	Retry Logon
		or: <CR><ESC><EOT><CR>	Forced disconnect by paging service. The BMC firmware should drop the transmission on the <EOT> and not require a <CR> to follow the <EOT>.
6a		(optional) <Message Sequence> <CR>	The paging service may insert a greeting message sequence at this point.
7		<ESC>[p<CR>	The paging service is ready to receive the first transaction. Note that the 'p' is in lowercase. This response should be received within t3 (10) seconds after step 6 / 6a.
8	<STX> Pager_ID<CR> Message<CR> <ETX> Checksum<CR>		The message transaction should be sent within t4 (4) seconds of getting the step 7 response from the paging service. Message data is transferred as a series of blocks, with message strings (fields) within a block. A field is a series of characters with <CR> indicating end-of-field. A block begins with <STX> and ends with a TAP checksum followed by a <CR>. The last character before the checksum is set to either <ETB> if the paging transaction takes multiple blocks, <US> if a field spans blocks, or <ETX> if the transaction ends within the block. The pages sent by IPMI consist of a single block with two fields: Field 1 : Pager ID (a.k.a. PIN) Field 2 : Message This format is shown in the Remote Entry Device column. The Pager ID number is determined by the paging service. Most Pager IDs are 7 ASCII numeric characters long. Note

			that there's no restriction in the spec on the maximum length for a Pager ID, nor any restriction that the paging service only require numeric characters. Any characters in the 7-bit ASCII set, including control characters, may be required. The response to each block (step 9) should be sent by the paging service within t3 (10) seconds of receiving the block. If the response is not received, the block may be resent. This can be repeated up to n2 (3) times before the BMC gives up and drops the line.
9		<Message Sequence> followed by either:	The specification indicates that there should be a message sequence from the paging service at this point, which should be a 3-character response code. However, it's possible that some paging service implementations will skip sending it. The BMC thus should accept an ACK/ NAK/ Reject/ or EOT sequence without a preceding message sequence. Per the TAP spec, older versions of TAP did not send a message sequence, and earlier implementations may just send <Ctrl-code><CR> instead of <CR><Ctrl-code><CR>. It's recommended that the BMC accept either the 2- or 3-character versions of the ACK/ NAK/ Reject/ or EOT/ sequences in order to provide a level of backward compatibility with earlier TAP services.
		<CR><ACK><CR>	Message acknowledged, ready for next block (if any).
		or: <CR><NAK><CR>	Message NAK'd (not acknowledged). Typically due to a checksum or transmission error. Resend block.
		or: <CR><RS><CR>	Transaction rejected. Don't retry. Transaction violates TAP protocol, field contents (e.g. Pager ID) were invalid, or other issue with the paging service. The BMC should give up and terminate the page at this point. It's probably best to follow the TAP spec and send an <EOT><CR> and wait for acknowledge from the paging service rather than just dropping the phone connection.
		or: <CR><ESC><EOT><CR>	Forced disconnect by paging service. The BMC firmware should drop the transmission on the <EOT> and not require a <CR> to follow the <EOT>.
10	<EOT> <CR>		Sent to paging service to indicate the end of the paging transactions.
11a		(optional) <Message Sequence><CR>	The paging service may insert a termination message sequence at this

			point.
11b		(optional) <RS><CR>	The paging service may indicate a transaction reject at this point.
11c		<ESC><EOT><CR> drop connection & hang up.	The paging service should indicate the end of the transaction by transmitting this sequence and hanging up at this point. While the specification doesn't state this, it's recommended that the BMC resend <EOT><CR> at "t6"* (2) "n4"* (3) times until acknowledged. The reason for this provision is to help ensure that the page transaction is accepted. Regardless, the BMC should time out after t3 (10) seconds after sending the last <EOT><CR> and drop the connection.

* t6 and n4 are new timing /retry parameters just associated with the IPMI specification.

Appendix G - Command Assignments

The following lists the commands defined in this specification and the minimum privilege level required to execute a given command. In addition, the following apply:

- Unless otherwise specified, unauthenticated, session-less interfaces, such as the System Interface and IPMB, can support any IPMI command.
- The privilege level requirements for OEM commands (NetFn=OEM, OEM/Group) is specified by the OEM identified by the corresponding manufacturer ID.
- Note that the *Send Message* and *Master Write-Read* commands are not available at the User privilege level, with the exception of using a *Send Message* command to deliver a message to the System Interface. This is because these commands enable unfiltered access the IPMB, ICMB, private management busses, and PCI Management Bus. This would potentially allow someone to use those commands to send commands to other controllers or write to non-intelligent devices on those busses. As a consequence, a User is only able to read FRU and sensors directly managed by the BMC. In addition, FRU must be accessed via the *Read FRU* command and not *Master Write-Read*.
- The *Send Message* command can be used to deliver a message to the System Interface at User privilege level. It is up to the system software to determine the privilege level and place any additional restrictions on messages received via the Receive Message Queue. This can be accomplished by using the session handle associated with the message and the *Get Session Info* command to look up the privilege level that the user is operating at. Software can also check the limits for the channel and the user by using information from the *Get Channel Access* and *Get User Access* commands to determine whether a given user has sufficient privilege to deliver a particular command to system software.
- Unless otherwise specified, the listed IPMI commands, if supported, must be accessible via LUN 00b.

Key for Command Privilege Levels Table:

b = command only generated by BMC, can be sent prior to a session being established
p = works at any privilege level, can be sent prior to a session being established
s = command executable via system interface only
X = supported at given privilege level or higher
I = command executable from local interfaces only (e.g. IPMB, SMBus, PCI Mgmt. bus or System Interface)
C = Callback privilege
U = User Privilege level
O = Operator Privilege level
A = Administrator Privilege level
App = Application Network Function Code
S/E = Sensor/Event Network Function Code
- = Reserved/unassigned, or OEM specified

Table G-1, Command Number Assignments and Privilege Levels

	section	NetFn	CMD	C	U	O	A
IPM Device "Global" Commands							
reserved	-	App	00h	-	-	-	-
Get Device ID	17.1	App	01h		X		
Broadcast 'Get Device ID'[1]	17.9	App	01h	I	I	I	I
Cold Reset	17.2	App	02h				X
Warm Reset	17.3	App	03h				X
Get Self Test Results	17.4	App	04h		X		
Manufacturing Test On	17.5	App	05h				X
Set ACPI Power State	17.6	App	06h				X
Get ACPI Power State	17.7	App	07h		X		
Get Device GUID	17.8	App	08h		X		
reserved	-	App	09h-0Fh	-	-	-	-
BMC Watchdog Timer Commands							
Reset Watchdog Timer	21.5	App	22h			X	
Set Watchdog Timer	21.6	App	24h			X	
Get Watchdog Timer	21.7	App	25h		X		
BMC Device and Messaging Commands							
Set BMC Global Enables	18.1	App	2Eh	s	s	s	s
Get BMC Global Enables	18.2	App	2Fh		X		
Clear Message Flags	18.3	App	30h	s	s	s	s
Get Message Flags	18.4	App	31h	s	s	s	s
Enable Message Channel Receive	18.5	App	32h	s	s	s	s
Get Message	18.6	App	33h	s	s	s	s
Send Message	18.7	App	34h		X[2]	X	
Read Event Message Buffer	18.8	App	35h	s	s	s	s
Get BT Interface Capabilities	18.9	App	36h		X		
Get System GUID	18.13	App	37h	p[3]	p[3]	p[3]	p[3]
Get Channel Authentication Capabilities	18.12	App	38h	p[3]	p[3]	p[3]	p[3]
Get Session Challenge	18.14	App	39h	p[3]	p[3]	p[3]	p[3]
Activate Session	18.15	App	3Ah	p[3]	p[3]	p[3]	p[3]
Set Session Privilege Level	18.16	App	3Bh		X[4]		
Close Session	18.17	App	3Ch	X[5]			
Get Session Info	18.18	App	3Dh		X		
unassigned	-	App	3Eh	-	-	-	-
Get AuthCode	18.19	App	3Fh			X	
Set Channel Access	18.20	App	40h				X
Get Channel Access	18.21	App	41h		X		
Get Channel Info Command	18.22	App	42h		X		
Set User Access Command	18.23	App	43h				X
Get User Access Command	18.24	App	44h			X	
Set User Name	18.25	App	45h				X
Get User Name Command	18.26	App	46h			X	
Set User Password Command	18.27	App	47h				X
unassigned	-	App	48h-51h	-	-	-	-
Master Write-Read	18.10	App	52h			X	
Chassis Device Commands							
Get Chassis Capabilities	22.1	Chassis	00h		X		
Get Chassis Status	22.2	Chassis	01h		X		
Chassis Control	22.3	Chassis	02h			X	
Chassis Reset	22.4	Chassis	03h			X	
Chassis Identify	22.5	Chassis	04h			X	
Set Chassis Capabilities	22.6	Chassis	05h				X

	section	NetFn	CMD	C	U	O	A
Set Power Restore Policy	22.7	Chassis	06h			X	
Get System Restart Cause	22.9	Chassis	07h		X		
Set System Boot Options	22.10	Chassis	08h			X[6]	
Get System Boot Options	22.11	Chassis	09h			X	
unassigned	-	Chassis	0Ah-0Eh	-	-	-	-
Get POH Counter	22.12	Chassis	0Fh		X		
Event Commands							
Set Event Receiver	23.1	S/E	00h				X
Get Event Receiver	23.2	S/E	01h		X		
Platform Event (a.k.a. “Event Message”)	23.3	S/E	02h			X	
unassigned	-	S/E	03h-0Fh	-	-	-	-
PEF and Alerting Commands							
Get PEF Capabilities	24.1	S/E	10h		X		
Arm PEF Postpone Timer	24.2	S/E	11h				X
Set PEF Configuration Parameters	24.3	S/E	12h				X
Get PEF Configuration Parameters	24.4	S/E	13h			X	
Set Last Processed Event ID	24.5	S/E	14h				X
Get Last Processed Event ID	24.6	S/E	15h				X
Alert Immediate	24.7	S/E	16h				X
PET Acknowledge	24.8	S/E	17h	p	p	p	p
Sensor Device Commands							
Get Device SDR Info	29.2	S/E	20h	I	I	I	I
Get Device SDR	29.3	S/E	21h	I	I	I	I
Reserve Device SDR Repository	29.4	S/E	22h	I	I	I	I
Get Sensor Reading Factors	29.5	S/E	23h		X		
Set Sensor Hysteresis	29.6	S/E	24h			X	
Get Sensor Hysteresis	29.7	S/E	25h		X		
Set Sensor Threshold	29.8	S/E	26h			X	
Get Sensor Threshold	29.9	S/E	27h		X		
Set Sensor Event Enable	29.10	S/E	28h			X	
Get Sensor Event Enable	29.11	S/E	29h		X		
Re-arm Sensor Events	29.12	S/E	2Ah			X	
Get Sensor Event Status	29.13	S/E	2Bh		X		
Get Sensor Reading	29.14	S/E	2Dh		X		
Set Sensor Type	29.15	S/E	2Eh			X	
Get Sensor Type	29.16	S/E	2Fh		X		
FRU Device Commands							
Get FRU Inventory Area Info	28.1	Storage	10h		X		
Read FRU Data	28.2	Storage	11h		X		
Write FRU Data	28.3	Storage	12h			X	

	section	NetFn	CMD	C	U	O	A
SDR Device Commands							
Get SDR Repository Info	27.9	Storage	20h		X		
Get SDR Repository Allocation Info	27.10	Storage	21h		X		
Reserve SDR Repository	27.11	Storage	22h		X		
Get SDR	27.12	Storage	23h		X		
Add SDR	27.13	Storage	24h			X	
Partial Add SDR	27.14	Storage	25h			X	
Delete SDR	27.15	Storage	26h			X	
Clear SDR Repository	27.16	Storage	27h			X	
Get SDR Repository Time	27.17	Storage	28h		X		
Set SDR Repository Time	27.18	Storage	29h			X	
Enter SDR Repository Update Mode	27.19	Storage	2Ah			X	
Exit SDR Repository Update Mode	27.20	Storage	2Bh			X	
Run Initialization Agent	27.21	Storage	2Ch			X	
SEL Device Commands							
Get SEL Info	25.2	Storage	40h		X		
Get SEL Allocation Info	25.3	Storage	41h		X		
Reserve SEL	25.4	Storage	42h		X		
Get SEL Entry	25.5	Storage	43h		X		
Add SEL Entry	25.6	Storage	44h			X	
Partial Add SEL Entry	25.7	Storage	45h			X	
Delete SEL Entry	25.8	Storage	46h			X	
Clear SEL	25.9	Storage	47h			X	
Get SEL Time	25.10	Storage	48h		X		
Set SEL Time	25.11	Storage	49h			X	
Get Auxiliary Log Status	25.12	Storage	5Ah		X		
Set Auxiliary Log Status	25.13	Storage	5Bh				X
LAN Device Commands							
Set LAN Configuration Parameters	19.1	Transport	01h				X
Get LAN Configuration Parameters	19.2	Transport	02h			X	
Suspend BMC ARPs	19.3	Transport	03h				X
Get IP/UDP/RMCP Statistics	19.4	Transport	04h		X		
Serial/Modem Device Commands							
Set Serial/Modem Configuration	20.1	Transport	10h				X
Get Serial/Modem Configuration	20.2	Transport	11h			X	
Set Serial/Modem Mux	20.3	Transport	12h			X	
Get TAP Response Codes	20.4	Transport	13h		X		
Set PPP UDP Proxy Transmit Data	20.5	Transport	14h	s	s	s	s
Get PPP UDP Proxy Transmit Data	20.6	Transport	15h	s	s	s	s
Send PPP UDP Proxy Packet	20.7	Transport	16h	s	s	s	s
Get PPP UDP Proxy Receive Data	20.8	Transport	17h	s	s	s	s
Serial/Modem Connection Active	20.9	Transport	18h	b	b	b	b
Callback	20.10	Transport	19h		[7]		X'
Set User Callback Options	20.11	Transport	1Ah				X
Get User Callback Options	20.12	Transport	1Bh		X		
Bridge Management Commands (ICMB)							
Get Bridge State	[ICMB]	Bridge	00h		X		
Set Bridge State	[ICMB]	Bridge	01h			X	
Get ICMB Address	[ICMB]	Bridge	02h		X		
Set ICMB Address	[ICMB]	Bridge	03h			X	
Set Bridge ProxyAddress	[ICMB]	Bridge	04h			X	
Get Bridge Statistics	[ICMB]	Bridge	05h		X		
Get ICMB Capabilities	[ICMB]	Bridge	06h		X		
Clear Bridge Statistics	[ICMB]	Bridge	08h			X	

	section	NetFn	CMD	C	U	O	A
Get Bridge Proxy Address	[ICMB]	Bridge	09h		X		
Get ICMB Connector Info	[ICMB]	Bridge	0Ah		X		
Get ICMB Connection ID	[ICMB]	Bridge	0Bh		X		
Send ICMB Connection ID	[ICMB]	Bridge	0Ch		X		
Discovery Commands (ICMB)							
PrepareForDiscovery	[ICMB]	Bridge	10h			X	
GetAddresses	[ICMB]	Bridge	11h		X		
SetDiscovered	[ICMB]	Bridge	12h			X	
GetChassisDeviceId	[ICMB]	Bridge	13h		X		
SetChassisDeviceId	[ICMB]	Bridge	14h			X	
Bridging Commands (ICMB)[8]							
BridgeRequest	[ICMB]	Bridge	20h			X	
BridgeMessage	[ICMB]	Bridge	21h			X	
Event Commands (ICMB) [8]							
GetEventCount	[ICMB]	Bridge	30h		X		
SetEventDestination	[ICMB]	Bridge	31h			X	
SetEventReceptionState	[ICMB]	Bridge	32h			X	
SendICMBEventMessage	[ICMB]	Bridge	33h			X	
GetEventDestination (optional)	[ICMB]	Bridge	34h		X		
GetEventReceptionState (optional)	[ICMB]	Bridge	35h		X		
OEM Commands for Bridge NetFn							
OEM Commands	[ICMB]	Bridge	C0h-FEh	-	-	-	-
Other Bridge Commands							
Error Report (optional)	[ICMB]	Bridge	FFh		X		

1. This command is sent using the Broadcast format on IPMB. See command description for details.
2. A User can use a *Send Message* command to deliver a message to system software, but Operator privilege is required to use it to access other channels.
3. Command only applies to authenticated channels.
4. This is effectively a no-op if the user has a maximum privilege limit of User since the command could not be used to change the operating privilege level to a higher value.
5. A session operating at Callback, User, or Operator can only use this command to terminate their own session. An Administrator or system software can use the command to terminate any session.
6. There is a bit in this command that can only be set at Administrator privilege level.
7. Command available for all levels except for User level
8. See [ICMB] specification for command specifications.

Index

B

C

D

E

F

G

H

I

K

L

M

Q

R

S

V

W

X