- IPMI -

Intelligent Platform Management Bus Communications Protocol Specification

v1.0

Document Revision 1.0

November 15, 1999

Intel Hewlett-Packard NEC Dell

Revision History

Date	Ver	Rev	Modifications
9/16/98	1.0	1.0	Initial release.
11/15/99	1.0	1.0	Updated license agreement.

LIMITED RELEASE SPECIFICATIONS LICENSE AGREEMENT

Intel Corporation; Hewlett Packard Company, NEC Corporation and Dell Computer Corporation (collectively "Licensor") through Intel Corporation (the "Administrator") will allow you to copy the Intelligent Platform Management Interface Specification (v1.0), Intelligent Platform Management Bus Bridge Specification (v 1.0), and Intelligent Chassis Management Bus Bridge Specification (v1.0) (the "Specifications") found at the IPMI website (the "Site") on the condition that you accept the terms and conditions below ("Agreement").

IMPORTANT - READ BEFORE DOWNLOADING OR COPYING. BY SELECTING THE "I ACCEPT" BUTTON BELOW, OR BY DOWNLOADING OR COPYING THE SPECIFICATIONS, YOU AGREE TO BE BOUND BY THE TERMS AND CONDITIONS STATED IN THIS AGREEMENT. IF YOU SELECT "I DO NOT ACCEPT," THE DOWNLOAD PROCESS WILL NOT PROCEED. DO NOT SELECT "I ACCEPT," DOWNLOAD OR COPY THIS SPECIFICATIONS UNTIL YOU HAVE CAREFULLY READ, UNDERSTOOD AND AGREED TO THE FOLLOWING TERMS AND CONDITIONS. IF YOU DO NOT WISH TO AGREE TO THESE TERMS AND CONDITIONS DO NOT DOWNLOAD OR COPY THE SPECIFICATIONS.

LICENSE. You may download the Specifications from the Site and make copies of the Specifications subject to these conditions:

1. You may not copy, modify, reproduce, disclose, rent, sell, distribute, transmit or transfer all or any part of the Specifications except as provided in this Agreement, and you agree to use reasonable efforts to prevent such actions for any copy of the Specifications that you have received subject to this Agreement.
2. **You may make copies of the Specifications for your review only and not for implementation. Before implementing the specifications, you must enter into an Adopters Agreement.** Information about the Adopters Agreement and the Adopters Agreement itself can be found at the IPMI website.
3. You may not remove or alter the disclaimers or license which is included in the Specifications on any copies you make.

NO OTHER LICENSE. Implementations developed using the information provided in the Specifications might infringe the intellectual property rights of various parties including the Licensor, other Adopters and other parties involved in the development of the Specifications. Except as expressly set forth in an Adopters Agreement which has been executed by You and accepted and executed by the Administrator, no license or right is granted to you, by implication, estoppel, or otherwise, under any patents, copyrights, maskworks, trade secrets, or other intellectual property by virtue of entering into this Agreement, downloading the Specifications, using the Specifications or building products complying with the Specifications.

OWNERSHIP OF SPECIFICATIONS AND COPYRIGHTS. Title to all copies of the Specifications remain with Licensor. The Specifications are copyrighted and are protected by the laws of the United States and other countries, and international treaty provisions. You may not remove any copyright or other proprietary rights notices from the Specifications. Licensor may make changes to the Specifications, or to items referenced therein, at any time without notice. Licensor is not obligated to support or update the Specifications.

PERFORMANCE INFORMATION. Performance information contained in the Specifications is intended to demonstrate potential performance characteristics or tradeoffs in implementing to the Specifications. Tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

EXCLUSION OF WARRANTIES. THE SPECIFICATIONS ARE PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING WITHOUT LIMITATION ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Licensor disclaims all liability, direct or indirect, for any claim relating to the Specifications or the use of information therein including without limitation claims arising from product liability, personal injury, death, or infringement of any proprietary rights.

THE SPECIFICATIONS ARE NOT LICENSED FOR USE BY, OR INTENDED TO DIRECT OR INSTRUCT, ANY PARTY IN THE DEVELOPMENT OF ANY IMPLEMENTATION WHERE FAILURE OF THE IMPLEMENTATION COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF USE, DIRECT, INCIDENTAL, CONSEQUENTIAL, OR SPECIAL DAMAGES, REGARDLESS OF WHETHER LICENSOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Nothing in this Agreement shall be construed as a sale or an offer for sale or license of any product.

TERMINATION OF THIS AGREEMENT. You may terminate this Agreement at any time upon written notice. If you breach this Agreement, Licensor may terminate this Agreement at any time upon written notice. Upon termination, you will immediately destroy the Specifications or return all copies of the Specifications to the Administrator and certify in writing to the Administrator that all your copies of the Specifications have been returned or destroyed.

APPLICABLE LAWS. Claims arising under this Agreement shall be governed by the laws of Delaware, without regard to principles of conflict of laws. You may not export the Specifications in violation of applicable export laws and regulations.

GOVERNMENT RESTRICTED RIGHTS. The Specifications and documentation are provided with "RESTRICTED RIGHTS." Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013, et seq. Use of the Specifications by the Government constitutes acknowledgment of Licensor's proprietary rights in them. Contractor or manufacturer is Licensor as identified on the Site.

Table of Contents

List of Figures

List of Tables

1. Introduction

The *Intelligent Platform Management Bus Communications Protocol Specification v1.0* defines a byte-level transport for transferring *Intelligent Platform Management Interface Specification* (IPMI) messages between intelligent I^2C devices. This protocol is layered above the bit stream and hardware interface defined by the 100 kbps I^2C specification. Together, the protocol and I^2C physical interface form the Intelligent Platform Management Bus.

The intelligent devices using the Intelligent Management Bus are typically microcontrollers that perform platform management functions such as servicing the front panel interface, monitoring the baseboard, hot-swapping disk drives in the system chassis, etc. Intelligent devices are also referred to in this document as 'management controllers' or just 'controllers'.

The information contained within is organized as follows:

Chapter 1: Introduction
Describes this document, introduces the protocol/service model, and provides background information.

Chapter 2: Functional Specification
Describes the request/response mechanism, link layer addressing, message transaction protocol, and network functions.

Chapter 3: Network Functions
Describes the network functions and the types of commands they support.

Chapter 4: Timing Specifications
Presents the timing requirements and limit counts for implementations of the protocol.

Chapter 5: Example Message Formats
Presents examples of messages formatted according to this specification.

1.1 Audience

This document is written for engineers and system integrators involved in the design and programming of systems and software that use the Intelligent Platform Management Bus to communicate between management controllers. It is assumed that the reader is familiar with PC and Intel Architecture servers, and microcontroller devices. For basic and/or supplemental information, refer to the appropriate reference documents.

1.2 Reference Documents

The following documents should be nearby when using this specification:

- *The I^2C Bus and How to Use It*, ©1995, Philips Semiconductors.
  Provides the specification for the I^2C bus, which defines bit-level interactions, electrical characteristics, and timing.
- *IPMB v1.0 Address Allocation*, ©1998, Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation.
- *Intelligent Platform Management Interface Specification v1.0*, ©1998, Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Computer Corporation.
  Provides the specifications of the command codes and data fields used for Intelligent Platform Management devices that reside on the IPMB. This includes the specification of requests and responses implemented for the configuration and access of platform event generation and sensor functions. The specified commands are also applicable when accessing Intelligent Platform Management devices via other messaging interfaces, such as the Intelligent Chassis Management Bus.

1.3 Conventions, Terminology, and Notation

The following table lists common terms and abbreviations used in this document.

Table 1-1, Glossary

Term	Definition
ACK	Acknowledge. The I^2C specification defines an extra clock pulse after each data byte during which the I^2C slave's acknowledges data received from the I^2C master (master write operation). If the master is receiving data from the slave (master read operation), the master sends an ACK to the slave after each byte. The master signals the slave to stop transmitting by not sending an ACK on the last byte. This is also called a 'NAK'.
BMC	Baseboard Management Controller.
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.
FPC	Front Panel Controller.
FRU	Field Replaceable Unit. A unit that can be readily replaced in the field to effect the repair of a system.
Hard Reset	A reset event in the system that initializes all components and invalidates caches.
HSC	Hot-Swap Controller.
I^2C bus	Inter Integrated Circuit bus. Simple 2-wire bi-directional serial bus developed by Philips for an independent communications path between embedded ICs on printed circuit boards and subsystems. The I^2C bus is used on and between system boards for internal system management and diagnostic functions.
ICMB	Term for an Intelligent Chassis Management Bus that runs 'outside the box' between separate physical chassis. Addressing on the Intelligent Chassis Management Bus is separate from the Intelligent Platform Management Bus.
IMB	Intelligent Management Bus. Abbreviation for the architecture and protocol used to interconnect intelligent controllers via an I^2C based serial bus for the purpose of platform management.
Internal (local) bus	Alternate term for the Intelligent Platform Management Bus connection that resides within a single enclosure. All devices on a local bus share the same internal 'I^2C' slave address space.
IPMB	Intelligent Platform Management Bus. Abbreviation for the architecture and protocol used to interconnect intelligent controllers via an I^2C based serial bus for the purpose of platform management.
kbps	kilobits per second.
LRU	Least Recently Used. Name for algorithms where the least recently used item in a fixed size list is dropped when a new element needs to be added.
LUN	Logical Unit Number. In the context of the 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.
NAK	Not Acknowledge. The I^2C specification defines an extra clock pulse after each data byte during which the slave's data output is switched low to indicate proper reception of the byte. Otherwise, NAK is indicated, which means that the byte has been rejected (in the case of a master write to a slave) or is the last byte the master wants to read (in the case of a master read from a slave).
Node	An entity operating on a segment of the Intelligent Platform Management Bus. In the context of this document, nodes are typically implemented using microcontrollers with I^2C interfaces. .
PBC	Processor Board Controller.
PSC	Power Share Controller.
RAS	Reliability, Accessibility, Serviceability
rq	Abbreviation for 'Requester'.
rs	Abbreviation for 'Responder'.
Slave Address	I^2C term. Name for the upper 7-bits of the first byte of an I^2C transaction, used for selection of different devices on the I^2C bus. The least significant bit of this byte is a read/write direction bit. The entire first byte of the I^2C transaction is also sometimes referred to as the 'slave address' byte.

Table 1-2, Notation

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

1.4 Scope

The protocol defined in this document is the means by which an intelligent device communicates with another on the Intelligent Platform Management Bus. *The protocol specification **does not** apply to an intelligent device's communications with devices that do not implement the protocol, such as a non-intelligent I^2C slave that also resides on the bus.*

1.5 Background

The Intelligent Platform Management Bus is designed to be incorporated into mission-critical server platforms for the main purpose of supporting 'Server Platform Management'. The bus can also be used to support platform management functions within peripheral chassis, and in non-server systems.

Autonomous microcontrollers, also referred to in this document as management controllers, or 'nodes', collect status and event information about the system independent of system software. This includes information on board voltages, temperatures, fan rotation speed, processor and bus failures, FRU part numbers and serial numbers, etc. The goal of this is to improve the RAS characteristics of the system by providing information that can be used to tolerate, predict, identify, or recover from the failure of a FRU.

The Intelligent Platform Management Bus architecture and protocol addresses several goals:

- **Support a Distributed Management architecture:** The Intelligent Platform Management Bus supports a distributed platform management architecture. Sensors and controllers can be located on the managed modules and their information consolidated via the IPMB. This yields a much more flexible design than one where all the sensors have to be directly routed to a central point of management.

- **Support Asynchronous Event Notification and Critical Event Logging:** The IPMB implements a multi-master protocol that allows intelligent controllers to arbitrate the bus for the purpose of sending an 'Event Message' to an 'Event Receiver' node. This provides a mechanism whereby a controller can raise an asynchronous event.

  IPMI-compatible platforms incorporate an 'Event Receiver' node that passes the message contents on to a System Event Log function for logging. The Event Receiver may also pass the Event Message on to a system software function, such as an SMI Handler, for immediate processing.

- **Provide an Extensible Platform Management Infrastructure:** New management information sources can be readily added to the Intelligent Platform Management Bus without impacting other controllers on the bus. New management controllers can be readily added by the system vendor or by third parties. In addition to allowing new devices to be accessed on the bus, the System Event Log mechanism supports a 'generic' event message types that allow new events, such as OEM unique events, to be integrated into the System Event Log and event handling without requiring firmware modifications.

- **Multi-master Operation:** The Intelligent Platform Management Bus implements a multi-master operation to support the distributed management architecture, asynchronous event notification, and platform extensibility. The mechanism supports direct communication between any two intelligent devices on the bus.

- **Support Non-intelligent I^2C Devices:** The IPMB has been designed to allow non-intelligent I^2C devices to be co-resident on the Intelligent Platform Management Bus. 'Non-intelligent' I^2C-based devices, such as SEEPROMs and I/O ports, can be incorporated as part of the platform management system. Such devices can be accessed directly, or can be managed as devices that are 'owned' by an intelligent controller.

- **Support 'Out-of-Band' Access to Platform Management Information:** The Intelligent Platform Management Bus is separate from the system's processor and memory busses. As such, it remains available even if a failure prevents the system from running. It is possible for the Intelligent Platform Management Bus to be augmented by system management add-in cards, such as autonomous management cards that connect to the management bus and allow management data to be delivered to a remote console via a phone line or LAN connection.

- **Reduce system management cabling & cost:** The Intelligent Platform Management Bus provides an inexpensive, low-pin count, communication media for platform management information. Only two additional signal lines are needed to route the management bus between internal modules such as the Baseboard, Processor Modules, Front Panel, Memory Cards, Hot-swap Backplanes, Power Share Board, etc.

  Miscellaneous system cabling functions, such as the routing of fault signals between modules, can be replaced by using the Intelligent Platform Management Bus. A dedicated wire is typically only used to communicate a single piece of management information, whereas the Intelligent Platform Management Bus carries whole streams of data.

  Systems can use the Intelligent Platform Management Bus to communicate Power Fault information between a Power Share Board that senses the failure, and a 'Front Panel Controller' that drives a power fault light. Since the management bus is already routed between the devices, it was unnecessary to route a dedicated signal line between the two.

- **Provide a Route to Interchassis Management:** The protocol has been designed to support the bridging of the Intelligent Platform Management Bus between chassis. This is done via 'store-and-forward' type devices referred to as 'bridge nodes'. The internal management bus address spaces are isolated by bridge nodes so there is no concern about address conflicts between the internal nodes in one chassis and those in another.

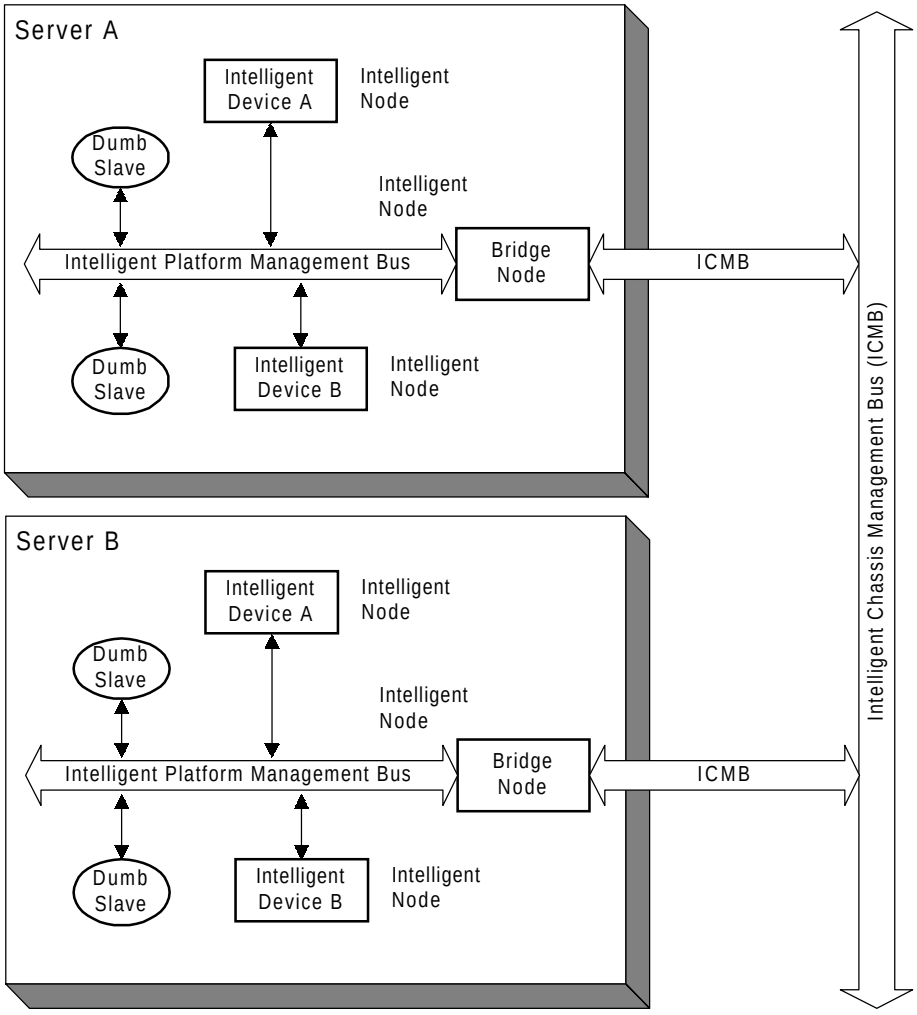
1.6 Platform Management Network Topology

Two classes of I^2C devices, non-intelligent and intelligent, are considered by this protocol. A typical non-intelligent device is a temperature sensor that indicates the current reading via I^2C. A typical intelligent device is the management controller on the server baseboard. Within the intelligent device resides firmware that implements the protocol along with other baseboard-specific functions. Non-intelligent devices typically have slave-only, device-specific, register-based interfaces, and lack the ability to implement the IPMB messaging protocol.

In a typical server, a number of intelligent and non-intelligent devices can be present on various modules in the system, such as the baseboard, processor/memory board, and server chassis. These devices operate as agents on the internal Intelligent Platform Management Bus segment. Each intelligent device is referred to as a node on the segment.

A similar approach is used for passing messages between systems using the ICMB protocol. To participate in ICMB communication, a server chassis provides an intelligent device that acts as the bridge node. A bridge node contains Intelligent Platform Management Bus ports for connection between the IPMB and the ICMB. Only intelligent devices, typically bridge nodes, reside on the Intelligent Chassis Management Bus, through which multiple servers are joined into a single Intelligent Chassis Management Bus domain. The following figure illustrates the IPMB and ICMB interconnection topology. Refer to the *Intelligent Chassis Management Bus Protocol Specification* for more information.

Figure 1-1, Intelligent Platform Management Bus Interconnection Topology



1.7 Application of the Protocol

The protocol concerns itself only with the transfer of data between intelligent nodes in an interconnected domain of the topology described below. With the exception of the generic request/response mechanism specified later in this document, how transferred data is used by intelligent devices is beyond the scope of this document. Refer to the *Intelligent Platform Management Interface Specification* for the definition and functional specifications of the message command sets.

1.7.1 Scale

While not a limitation of the communications protocol, the Intelligent Platform Management Bus has been targeted to directly support systems with up to 15 internal nodes. System implementation should strive to keep management bus occupancy light, with a target of fewer than 6 messages per second typically encountered on a typical IPMB bus. This is to ensure that nodes can successfully arbitrate for and access the bus under the recommended number of retries.

1.8 Request / Response Protocol

The IPMB uses a Request / Response protocol. A Request Message is issued to an intelligent device. The device responds with a separate Response Message. Both Request Messages and Response Messages are transmitted on the bus using I^2C Master Write transfers. That is, a Request Message is issued from an intelligent device acting as an I^2C master, and is received by an Intelligent Device as an I^2C slave. The corresponding Response Message is issued from the responding intelligent device as an I^2C master, and is received by the request originator as an I^2C slave.

1.9 Service Model Definition

The protocol defines how an intelligent device sends messages of various byte formats from node-to-node. Data transfer occurs synchronously with the invocation of this service, providing an indication to the source as to whether or not the transfer successfully occurred.

The physical management bus interface uses an 'ACK' bit to acknowledge bytes as they are received. A negative-acknowledge conveys rejection of the byte by the receiver. Receiving 'ACKs' for all bytes in a message transfer does not imply that the application within the addressed node has correctly parsed or handled the message. An I^2C bus acknowledge only conveys acceptance of the byte by the receiver.

Acceptance of request messages by the application within the node is indicated by a response message from the node. The response message contains information that indicates whether the request was accepted or rejected. Response messages themselves are not confirmed, other than by their bytes being acknowledged at the physical level.

The protocol also provides a broadcast message capability to allow the population of intelligent devices to be discovered so communication to those devices can be set up.

1.10 Design Objectives

The basic design of the protocol conforms to the following objectives and principles:

- Design for efficient implementation by microcontrollers with limited processing capability and limited RAM and ROM resources.
- Exhibit a high degree of symmetry, for example: the method of accessing an intelligent node should be independent from the node's function (i.e. whether the device is a front panel controller, power share board controller, or baseboard management controller).
- Exhibit a high degree of self similarity, for example: chassis addressing should be as similar in nature to base I^2C addressing as possible.
- Minimize the processing and memory burdens of bridging functionality.
- Exhibit clean layering of functionality, for example: messages (application data) are not mixed with protocol data, and different aspects of protocol data are similarly kept separate.

1.11 Protocol Use of I^2C Services

An I^2C bus is used as the media and physical layer for the Intelligent Platform Management Bus.

This protocol is restricted to the following use of I^2C services:

1. Only Master Write transactions are used (R/W bit on I^2C slave address is always 0).
2. 10-bit addressing is not used.
3. Only one slave address is used in a single transaction.
4. No repeated starts are used. Repeated starts are defined in the I^2C specification as a means of changing bus direction during a transfer. Refer to the references on I^2C for more information. Since the protocol only uses Master Write transactions, repeated starts are not required. Note, that while repeated starts are not used in Intelligent Platform Management Bus transactions they can be used for non-protocol transactions on the same bus. I.e. you can still use repeated starts to access devices that don't use the IPMB protocol.
5. The I^2C acknowledgment mechanism ('ACK' bit) only indicates acceptance of the byte by a slave and does not convey any additional information regarding the integrity or correctness of the received data.

2. Functional Specification

This chapter describes the form and function of various aspects of the protocol:

- Request/response mechanism
- Link layer addressing
- Message transaction protocol
- Network functions

2.1 Request / Response Mechanism

Each intelligent node has its own set of requests that it accepts, and responses that it generates. Their exact definitions are provided in the interface specification documents for each intelligent. However, the sets of common requests/responses are provided in the *Intelligent Platform Management Interface Specification*.

Intelligent devices on the Intelligent Platform Management Bus transmit requests and responses as bus *Masters* and receive requests and response as bus *Slaves*. Thus, the only Master Write transactions are seen between intelligent nodes.

2.2 Terminology

Prior to further discussion, some terms should be defined. A node issuing a request is referred to as the Requester; a node issuing a response as the Responder. For a given pair of nodes, each may over time assume both roles; however, for the sake of this discussion, assumption of single, complementary roles is assumed. The following table defines additional terminology used in descriptions of the protocol:

Table 2-1, Protocol Element Definitions and Notation

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

2.3 Directing responses back to the Requester

Intrinsic to the nature of a request is the requirement for a method of directing the response back to the Requester. The Requester provides this direction by supplying its **Requester's Slave Address (rqSA)** and **Requester's LUN (rqLUN)** in the request. This information is extracted and used to address the response back to the Requester.

2.4 Telling Requests from Responses

Requests and responses have different values for their network functions. Requests use even-numbered network function values and responses use the corresponding odd-numbered network function values. Thus, requests can be differentiated from responses by examining the netFn field.

Since it is possible that more than one request may be outstanding, it is important to confirm that a response is for a particular request. This is done using the following mechanisms:

1. The **Responder's Slave Address (rsSA)** and **Responder's LUN (rsLUN)** is included in the response; this tells the Requester which node the response came from.

2. The **Command (cmd)** field from the request is included in the response; this allows the Requester to verify that the response is for a particular outstanding request.

3. The **Seq** field from the request is returned in the response. This allows the Requester to verify which instance of a request the response is for.

2.5 Responding to Requests

The interface specification for each Responder defines the number of command bytes it implements. A Responder must provide at least one command byte in the specified field location. Any additional parameter bytes associated with the command field must immediately follow the first byte. Applications that issue requests to a Responder must be designed so that they can identify responses by parsing of the command field returned in the response.

Responses must be provided for all VALID requests. A valid request is defined as a request that uses a cmd, netFn, and LUN supported in the command set of the node, and passes the data integrity checks (checksum). The exception to this requirement occurs if the node receiving the request is in the middle or processing a request, or is waiting for a response from another node. In this case, the node can elect to NAK the request and skip providing a response. Note the preferred implementation is for the node to respond to the unexpected request by formatting a corresponding response message with a Completion Code of C0h = "Busy".

Responders are not required to queue incoming messages from a given Requester, or queue outgoing responses to a given Requester. That is, the interface can be 'single threaded'. A Responder that accepts an incoming request from a given Requester may flush any pending responses to that Requester. This operation is allowed in order to reduce the burden of implementation of the protocol in 'small controllers'.

Nodes should strive to preserve temporal ordering on the bus when possible: i.e. responses should be returned in the order that the corresponding requests were received by the Responder. Note that bridge nodes can be designed to allow multiple requests to be pending to multiple chassis. Such bridge nodes

will likely return responses to the IPMB in the order that the bridge node receives them on the ICMB. In this case, the temporal ordering would not be preserved.

2.5.1 Corrupted Request Handling

If a request is received with a bad checksum, it is questionable whether the Requester's slave address is correct or not. The responder should not send back an 'error' completion code. Instead, the responder should ignore the request entirely and let the requester retry the message..

2.6 Missing Response Handling (retries)

Missing responses are intended to be handled by the following mechanism:

1. The Requester times out waiting for the response.
2. The Requester retries the request and time-out until a retry count is exceeded.
3. The Requester can send a 'Get Device ID' command to see if the Responder is still operational. If this succeeds, then the Requester can send the global command for a 'Warm Reset' to the Responder as an attempt to return the Responder's IPMB communications interface to a known state. If this fails, the Requester may assume that the Responder has failed.

2.6.1 The Seq Field and Retries

The Requester changes the Seq field whenever it issues a new instance of a command (request). The Responder then returns the Seq field in the response, allowing the Requester to match the response with the given instance of the request.

The Requester should check the following fields (after verifying data integrity using the checksums) to match a response to a given request:

- rsSA
- rsLUN
- Seq
- netFn
- cmd

The Requester must change the Seq field value for each new instance of a request that has the same rsSA, rsLUN, netFn, and cmd fields as a previous request to the node (within the Seq field expiration interval, see Section 2.6.2, Seq Value Expiration). The value can be repeated among outstanding requests as long as at least one of the other fields is different.

A typical Requester implementation increments the Seq field for each new request. This is not mandatory. The Requester can change the Seq field in any sequence that meets the Requester's needs for response handling.

The Seq field is also used to manage retried Requests received at the Responder. If the Requester needs to retry a request, it re-issues the command without changing the Seq field value. The

Responder uses the Seq field to detect duplicate requests. A Responder that receives duplicate requests from a given Requester can elect to reject or accept duplicate requests based on whether there would be an undesirable side-effect or not. Thus, it is not mandatory for all Responder implementations to check for duplicate requests, or to check for duplicate requests on all commands.

For example, suppose a Responder accepts a request message to increment a counter. It would normally be undesirable for the counter to increment on a duplicate (retried) request. In this case, the Responder would reject the duplicate request. A different request message might be used return a system status. In this case, there's no side-effect and the Responder can elect to respond as if the duplicate request were a new request.

If the Responder accepts the duplicate request, it should respond as if the request were a new instance (if there's no side effect). If the Responder needs to reject the request, it should return either an appropriate non-zero completion code (generic Completion Code of CFh = "Cannot execute duplicated request" is recommended), or ignore the request and let the Requester retry. Refer to the *Intelligent Platform Management Interface Specification* for the list of Completion Codes.

2.6.2 Seq Value Expiration

If a Requester gets reset, it is possible that it would unintentionally produce a sequence number for a new request that matches the one that was last issued. This could be because of Seq value wrap-around, or because only one message has been sent from the Requester. In order to guard against this, it is recommended that sequence number expiration be implemented. Any request from a given Requester matches the previous request from a given requester but is received after the **'**Seq value expiration interval**'** should be treated as a new request, not as a retry. *Refer to the Timing Specifications section for the expiration interval.*

2.7 Response Time-outs

The Requester should not assume that it will successfully receive responses for all requests. It is possible for certain conditions to occur where a Responder may not provide a response. For example, it is possible that the address header of the request message could have been corrupted, or an invalid request was sent to a Responder that NAKs new requests while processing a request, or another request message, such as a 'Cold Reset' message, caused the Responder to skip sending the response.

The Requester should time out if a response is not received. *Refer to the section on Timing Specifications for time-out and retry requirements.* It is highly recommended that intelligent nodes also implement request retries. The specified number of retries is believed to be sufficient to provide reliable delivery on typical implementations of the Intelligent Platform Management Bus. Implementations can elect to raise the number of retries if required.

2.8 Unexpected Request Messages

Requests and responses are not atomically paired transactions. Other bus traffic is allowed in the interval between a given request and the expected response. Thus, intelligent nodes must allow for 'unexpected' messages to be received between issuing a request and getting a response. This requires nodes to check the addressing, network function, Seq, and command fields to verify that the received information is actually the expected response. (Refer to later sections of this document for message field definitions).

For example, node 'A' may issue a request to node 'B', and while waiting for the response node 'A' may itself get a request from node 'C'. A node can handle an 'unexpected request message' in several ways. First, the node can accept the message and process it. This is the most desirable method. If this cannot be done, the node can elect to just ignore the unexpected message.

2.9 Bad and Unexpected Response Messages

Unexpected response messages are simpler to handle. Any response message that a node receives that does not match up with a pending request from that node is ignored. Any response message that is expected but is corrupted or does not fully comply with the specifications of the protocol can be ignored.

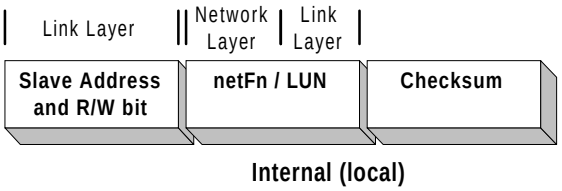
2.10 Link Layer Addressing

Any intelligent node can access another on the Intelligent Platform Management Bus using a link layer addressing sequence. Both point-to-point and broadcast addressing is supported. The link layer header is defined in terms of the bus across which the access is made: local (internal) or interchassis (external). Standard I^2C addressing (as defined in the I^2C specification) is used on all internal transactions.

2.10.1 Connection Header

This combination of link and network layers is referred to as the connection header. Successful transmission of a connection header establishes communications between nodes, thus preparing the way for the transmission of subsequent bytes that comprise the body of a message. The following figure shows the connection headers for internal and external accesses.

Figure 2-1, IPMB Connection Header Format



The connection header contains a 7-bit slave address followed by the I^2C read/write bit. Since the protocol uses only I^2C Master Write transfers , this read/write bit is always 0, indicating a write transfer. This byte is followed by the network function / LUN byte and a checksum byte. The checksum byte allows the integrity of the connection header to be verified. A node can reject the remaining bytes in the message if the connection header's checksum does not verify.

2.10.2 Non-intelligent Device Slave Addressing

Non-intelligent slave devices can reside on the same Intelligent Platform Management Bus segment as Nodes that implement the Intelligent Platform Management Bus protocol. Addressing conflicts are avoided at the I^2C bus level by assigning unique slave addresses to all devices on the management bus. Refer to the references on I^2C for more information.

The IPMB can also support SMBus slave devices, with the restrictions that the SMBus 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. The latter restriction is due to an overlap between the definitions of the 2nd byte in the message. In the SMBus Modified Write Word protocol, the second byte is defined as the originator's device address, while the 2nd byte in an IPMB message holds the NetFn & LUN fields.

2.10.3 Slave Address Specification

The definition of explicit slave addresses is beyond the scope of this document. The slave addresses for a particular implementation of the Intelligent Platform Management Bus are defined separately. Refer to the reference document *I^2C Address Allocation* for a specification of the address allocation used on Intel Architecture server platforms.

2.11 Message Transaction Formats

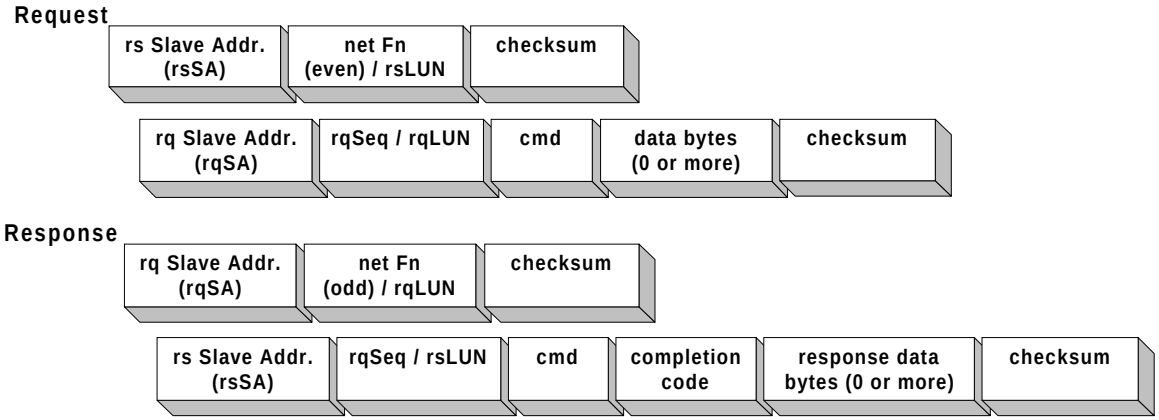
Requests use connection headers with an even network function, and responses use odd network function values. A given network function may require a specific command byte and other information. A response may also contain a string of data bytes, specific to the network function. The standard command and data byte specifications are given in the *Intelligent Platform Management Specification.* Network Function values can be found in *Section 3, Network Functions and Commands.*

In the figures that follow, the blocks represent the various fields of the message. Unless otherwise noted, these blocks also correspond to single bytes.

2.11.1 IPMB Node to Node Transactions

The following figure shows the message protocol for a node to node transaction on the IPMB. This consists of IPMB connection headers and appropriate command and data bytes for the given network function.

Figure 2-2, IPMB Node to Node Message Format



3. Network Functions and Commands

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 presents an overview of the network function code values. 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 overriding specification of the Network Function codes and the format and function of the standard IPMI command sets is provided in the *Intelligent Platform Management Interface Specification*.

The rest of this chapter describes each of these network functions and their usage, except for application and firmware transfer request/response functions, which are beyond the scope of this document (refer to interface specifications for individual devices).

Table 3-1, Network Functions

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.
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.
0Ah-2Fh	Reserved	-	reserved (38 Network Functions)
30h-3Fh	OEM	-	Vendor specific (16 Network Functions)

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

3.1 Completion Codes

Per the *Intelligent Platform Management Interface Specification*, intelligent IPMB nodes return a Completion Code in all response messages. 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 are intended to be applied at the 'command' level. That is, they are responses to the interpretation of the command *after* it has been received and validated through the messaging interface.

Completion Code values are split into 'generic' and 'command-specific' ranges. All commands can return Generic Completion Codes. Commands that completed 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. Refer to the *Intelligent Platform Management Interface Specification* for the specifications of 'generic' and 'command-specific' Completion Code values.

3.2 Application Messages (netFn 06, 07)

The Application network function is used for commands that are related to the node's unique 'Application' functionality. Application Messages are formatted to follow the internal and interchassis request and response formats specified in this document.

Per the *Intelligent Platform Management Interface Specification*, the command/response message set for Application messages is split into two categories: 'global' and 'node (device) specific'. Global command and response messages have the same data formats and command values for all IPMB nodes.

Node-specific command and response messages are specified particular Interface Specifications for a node. This specification is typically made via a table specifying the cmd and associated data passed for requests and responses. The LUN(s) for which the message is valid are also specified.

3.3 Sensor/Event Messages (netFn 04, 05)

The Sensor/Event network function is used for commands that are related to the configuration and access of Sensors and the configuration and transmission of platform events.

3.3.1 Event Messages

Event messages report the occurrence of critical events in server management such as power supply over-voltage, over-temperature conditions, bus errors, etc.

The IPMI *Platform Event* command is used for Event Messages. The term 'Event Message' is actually a short-hand for 'Platform Event Request message'. The terms 'Event Request Message' and 'Event Response Message' may also used. These correspond to 'Platform Event Request Message' and 'Platform Event Response Message', respectively.

There are unique timing and retry requirements for Event Messages (*Platform Event* request messages) on the IPMB.

Event Messages are generated by Intelligent Platform Management Bus controllers and delivered to a particular node that functions as an 'Event Receiver'. The format of Event Messages contains information indicating the source of the message (via the rqSA field), and a variable length event data field. The operation of the Event Receiver, and the format and content of the event data fields, are described in the *Intelligent Platform Management Interface Specification.*

A controller that has sent a Event Message is responsible for ensuring its receipt by the Event Receiver. It is expected to do this by maintaining an 'unconfirmed' status for the message. The controller will retain this status, and periodically retry sending the event message until a Event Message response is received for that message, or the controller's retry limit is reached. Refer to *Section 4, Timing Specifications*, for time-out and retry count specifications.

The Seq field is particularly important for handling event messages. The Seq field allows the Event Receiver to discriminate whether the Event Message is for a new occurrence of a given event, or is a retransmission of a previous Event Message for that event. A Requester must ensure that Seq field values are different between successive transmission of the same command.

3.3.2 Sensor Messages

Sensor Messages are used for the configuration and access of the sensor functions in Intelligent Platform Management Bus controllers. For example, a common sensor function is platform voltage sensing. The Sensor Messages are include messages for accessing the present state of the sensor and for configuring the event generation parameters on the sensor.

The data fields for Sensor and Event Messages share a common specification of the parameters that indicate such elements as 'Sensor Type', 'Event Type', and 'Event Data'. This is linked with information in 'Sensor Data Records' which are used to describe what sensors are available in a platform, their type, location, event generation capabilities, etc. The *Intelligent Platform Management Interface Specification* provides the specification of the commands and data field values used in Sensor Messages.

3.4 Firmware Messages (netFn 07, 08)

Firmware messages follow the IPMB request and response message formats specified in this document. The 'Firmware' network function specifies that the commands and responses belong to the set of Firmware Messages. Since firmware upload/download capability can vary significantly from node-to-node, this functionality is also considered to be 'node specific'. The command values and data formats for the firmware functionality of a node are specified in that node's Interface Specification.

3.5 Storage Messages (netFn 0A, 0B)

Storage messages also follow the internal and interchassis request and response formats specified in this document. The 'Storage' network function specifies that the commands and responses are for functions related to non-volatile storage access and configuration. In particular, the storage functions are used for commands for reading and writing Sensor Data Records, FRU (Field Replaceable Unit) information (such as board serial numbers and part numbers), and System Event Log records. The specification of the commands and data field values in Storage Messages for IPMB devices is provided in the *Platform Sensor and Event Interface EPS.*

3.6 Bridge Messages (netFn 02, 03)

Bridge nodes reside between the Intelligent Platform Management Bus and external message transports, such as the Intelligent Chassis Management Bus (ICMB).

Bridge nodes receive request and response messages from one bus and reformats them for the other bus. A message to be bridged is typically embedded in a 'bridge' message generated from the bus that sources the message. The bridge node strips the message contents from the bridge message, adds the appropriate addressing header and checksum information, and reformats and re-frames the message for transmission on the desired bus.

Refer to the *Intelligent Chassis Management Bus Specification* for the specification of IPMB-ICMB bridge messages, and the operation of IPMB-ICMB bridging.

3.7 Chassis Device Messages (netFn 00, 01)

The Chassis network function is used for messages related to control and status functions that are common to server system chassis and peripheral chassis. This includes functions such as power and reset control, main power subsystem status, overall cooling status, chassis FRU information, etc. These functions can be thought of as being implemented by a Chassis Device node in the particular chassis. Refer to the *Intelligent Chassis Management Bus Specification* and the *Intelligent Platform Management Interface Specification* for additional information and specifications on Chassis Device messages and functionality.

4. Timing Specifications

Table 4-1, Timing Specifications

Internal Timing Specifications		min	max	
Overall Message Duration	T1	-	20 ms	
Time-out waiting for bus free	T2	60 ms	-	
Time-out waiting for a response, internal	T3	60 ms[1]	T6max[1]	
Time between Event Message Requests	T4	5 ms	-	
Request-to-Response time	T5	-	T6max-T1max-3ms[1]	This interval is measured from the end of the request transmission through the end of response transmission. (I^2C STOP to I^2C STOP)
Number of Request retries	C1	5[2]	-	recommended
Time between Request retries	T6	60 ms	250 ms	
Number of Event Message Request retries	C2	3	10	
Overall Message Byte Duration	T7	per I^2C spec	3 ms	
I^2C Clock Low hold	T8	per I^2C spec	3 ms	
Seq value expiration interval	T9	5 seconds[3]		
Lockout time processing request	T10	-	20 ms	recommended A node that NAKs new requests while processing a current request should complete the request and transmit the response within this interval. This interval begins after the request message has been received.

Notes:

General: Timing values apply to local node-to-node messages only.

1. Unless otherwise specified, this timing applies to the mandatory and optional commands specified in the *Intelligent Platform Management Interface Specification*. Refer to the *Intelligent Chassis Management Bus* specification for the timing specifications for Bridge commands used for IPMB-to-ICMB bridging. For controller-specific Application and Firmware commands, the Responder should attempt to meet this specification. In cases that it cannot, the interface specification for the Responder must clearly specify the 'Request to Response' time that was implemented. Because timing can vary according to command and controller, communication routines should be designed to support response timeouts and retry counts accordingly.

2. This is a recommended value only. The protocol does not require that non-Event Message requests be retried. The implementation of retries and the number used is based on the application's requirements for message delivery.

3. If a Requester is reset, it may produce the same sequence number for a request as one that was previously issued. In order to guard against this, it is recommended that sequence number expiration be implemented. Any request from a given Requester that is received more than T9 seconds after a previous, matching, request should be treated as a new request, not a retry.

5. Example Message Formats

This chapter shows further examples of the byte format of messages using the protocol. Bytes are numbered starting at 0, which is the first byte sent in a message transaction. A series of bytes is shown as m::n, where m and n are the starting and ending bytes.

5.1 Application Messages (netFn 06, 07)

Application messages are commonly encountered messages on the IPMB. They are used for the majority of communications between nodes on the internal management bus. This section presents an example of an internal request and response for an application message.

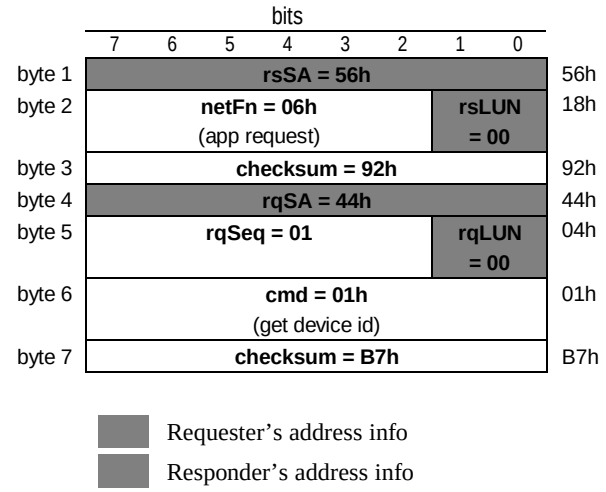
> *Note, the command formats shown are for illustration purposes only. Refer to the* Intelligent Platform Management Interface Specification*, or the controller's interface specification, for actual field specifications.*

For this example, suppose we are an intelligent node with slave address '44h' and LUN 00, and we wish to send a 'Get Device ID' request to the application functionality of LUN 00 of the internal node at slave address '56h'. It happens that this device returns 03h, 02h, 01h, 05h, 10h for it's device ID information. Let's also assume that this is the first time we've ever sent this command to this destination node, so we'll arbitrarily initialize our Requester's Seq to 01h. This yields the following data that will be used in the transaction:

Our slave address =	Requester's Slave Address (rqSA)	= 44h
Our LUN =	Requester's LUN (rqLUN)	= 00
Our Seq =	Requester's Seq (rqSeq)	= 01h
Target nodes' slave address =	Responder's Slave Address (rsSA)	= 56h
Target node's LUN =	Responder's LUN (rsLUN)	= 00
Command (cmd) =	Get Device ID	= 01h
Network Function (netFn) =	Application Request	= 06h
	Application Response	= 07h
Response completion code =	Application Response 'OK'	= 00h
Target node's Device ID info:	Device ID	= 03h
	Device Revision	= 02h
	Firmware Major Revision	= 01h
	Firmware Minor Revision	= 05h
	IPMI Spec Revision (1.0 → 0.1h)	= 01h

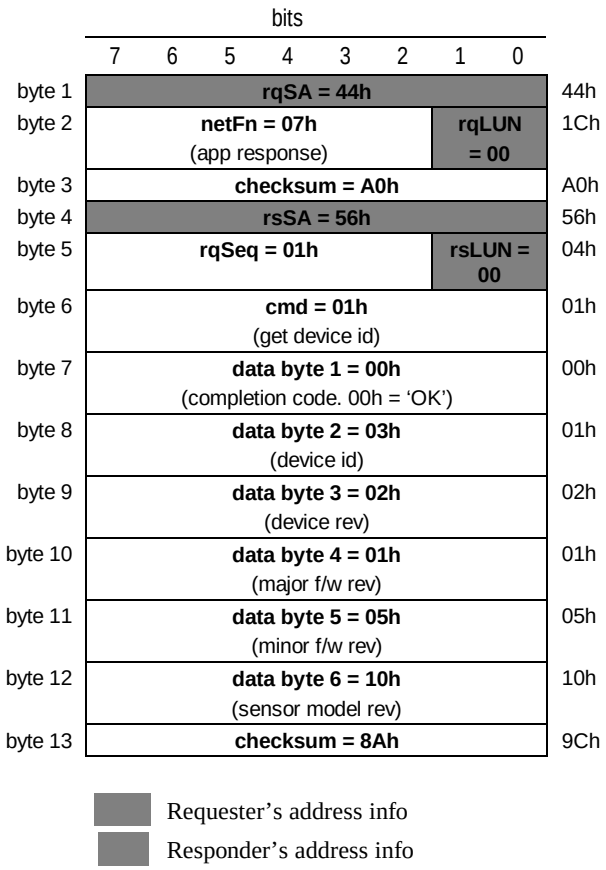
The corresponding internal application request message would be formatted as:

Figure 5-1, IPMB 'Get Device ID' Request



Assuming no errors occur, the Responder would provide the following response data:

Figure 5-2, IPMB 'Get Device ID' Response



6. Connectors

There are two types of standardized connector specified for providing auxiliary connection to the IPMB; the Type A connector, and the Type B connector.

The Type B connector is specified as providing access to a portion of the IPMB that remains active when the system is on standby power. Whereas the Type A connector may or may not be, dependent on the particular system implementation. Thus, devices that require access to the IPMB when the system is on standby power should attach to the Type B connector, if available. Otherwise, it is necessary to determine if the system has a Type A connector that provides standby access. (That information should be provided in the documentation for the system).

The following sections provide additional information and specifications on these connectors.

6.1 IPMB Connector, Type A

This 3-pin connector is used to provide a cabled baseboard connection for value-added features and 3rd-party add-in cards, such as Emergency Management Cards, that provide management features using the IPMB.

Devices that connect to a Type A connection that remains active on standby power must be designed such that they do not ground out the IPMB when the system is powered down.

The Type A connector is Molex part number 22-44-7031 or equivalent.

Devices that attach to the Type A connector must meet the loading specification in the following table. This corresponds to two I^2C standard-mode device loads. Remaining input and output voltage levels and current drive levels must meet the I^2C specification for standard-mode devices on a 5V I^2C bus.

Table 6-1, Type A Connector Device Loading

Parameter	Symbol	Min.	Max.	Unit
Input current, per pin (SDA, SCL)	I_i	-20	20	µA
Capacitance per pin (SDA, SCL)	C_i	-	20	pF

6.1.1 Pinout

Table 6-2, IPMB Connector, Type A - Pinout

PIN	Signal
1	IPMB_SDA
2	GND
3	IPMB_SCL

6.1.2 Dimensions and Layout

The following figures present the connector dimensions and recommended PCB layout. Note that the figures, below, are for a 5-pin connector. The actual connector uses a 3-pin version of the connector

with only three pin positions. I.e. the connector is *not* a five-pin connector with two empty pin positions.

Figure 6-1, IPMB Connector, Type A - Dimensions

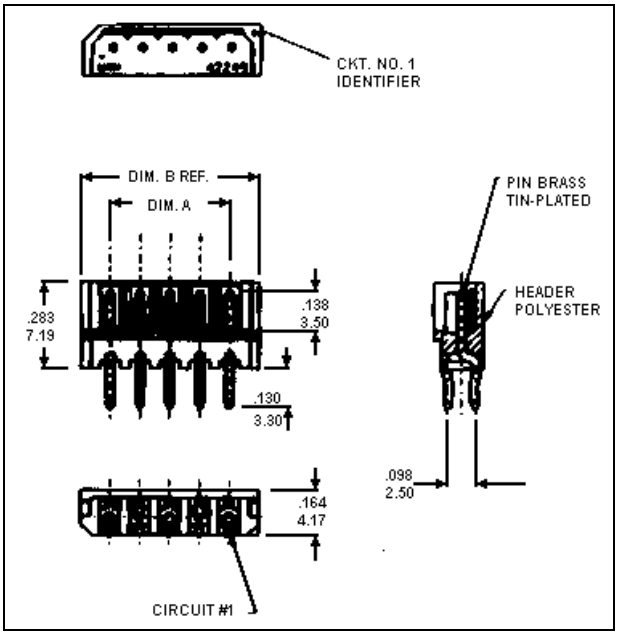
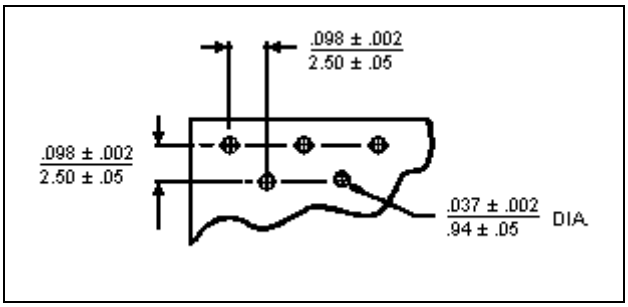


Figure 6-2, Aux. IPMB Connector Recommended PCB Layout



6.1.3 Electrical Requirements

Maximum Current Rating: 3 amps minimum continuous with proper mating connector.

Maximum Rated Voltage: 250 VAC RMS.

Dielectric Withstand: 1000 VAC RMS minimum when applied between adjacent terminals for 60 seconds.

Insulation Resistance: 1000 Mohm minimum when measured between adjacent pins.

6.1.4 Environmental Requirements

Operating Temperature Range: -40°C to +105°C.

6.2 IPMB Connector, Type B

This 4-pin connector is used to provide a cabled baseboard or front panel connection for value-added features and 3rd-party add-in cards, such as Emergency Management Cards, that provide management features using the IPMB.

The Type B connector must be used on a segment of the IPMB that remains operative when the system is powered down, and that provides access to the Chassis Device functionality of the system. This provides a standard mechanism that enables add-in cards to power up the system.

Devices that connect to the Type B connector must be designed such that they do not ground out the IPMB when the system is powered down.

The Type B connector is Molex part number 22-44-7041 or equivalent.

Devices that connect to the Type B must meet the loading specification in the following table. This corresponds to two I^2C standard-mode device loads. Remaining input and output voltage levels and current drive levels must meet the I^2C specification for standard-mode devices on a 5V I^2C bus.

Table 6-3, Type B Connector Device Loading

Parameter	Symbol	Min.	Max.	Unit
Input current, per pin (SDA, SCL)	I_i	-20	20	µA
Capacitance per pin (SDA, SCL)	C_i	-	20	pF

6.2.1 Pinout

On the add-in device, Pin 2 can be used as a "cable presence" input by connecting it to a pull-up resistor instead of to GND. Pin 4 is a "No connect". Its purpose is to help the user plug the add-on device into the correct IPMB segment.

Table 6-4, IPMB Connector, Type B - Pinout

PIN	Signal
Pin 1	IPMB_SDA
Pin 2	GND
Pin 3	IPMB_SCL
Pin 4	No connect

6.2.2 Dimensions and Layout

The following figures present the connector dimensions and recommended PCB layout. Note that the figures, below, are for a 5-pin connector. The actual connector uses a 4-pin version of the connector with only three pin positions. I.e. the connector is *not* a five-pin connector with an empty pin position.

Figure 6-3, IPMB Connector, Type B - Dimensions

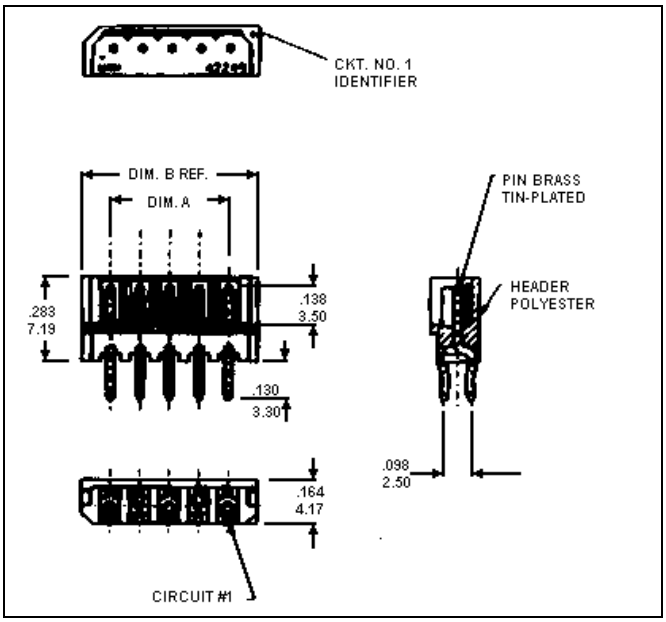
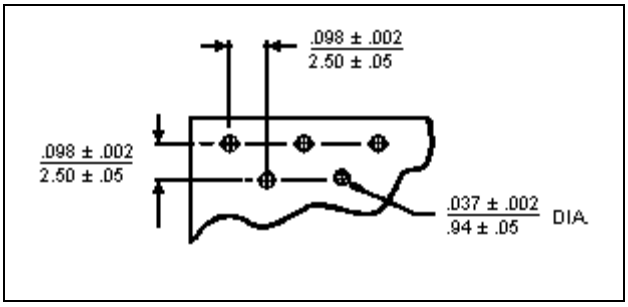


Figure 6-4, Aux. IPMB Connector Recommended PCB Layout



6.2.3 Electrical Requirements

Maximum Current Rating: 3 amps minimum continuous with proper mating connector.

Maximum Rated Voltage: 250 VAC RMS.

Dielectric Withstand: 1000 VAC RMS minimum when applied between adjacent terminals for 60 seconds.

Insulation Resistance: 1000 Mohm minimum when measured between adjacent pins.

6.2.4 Environmental Requirements

Operating Temperature Range: -40°C to +105°C.

LAST PAGE