Document Number: DSP0240

Date: 2009-04-23

Version: 1.0.0

Platform Level Data Model (PLDM) Base Specification

Document Type: Specification

Document Status: DMTF Standard

Document Language: E

CONTENTS

Figures

Tables

Foreword

The *Platform Level Data Model (PLDM) Base Specification* (DSP0240) was prepared by the Platform Management Components Intercommunications (PMCI) Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability.

Introduction

This document describes base protocol elements of the Platform Level Data Model (PLDM) for the purpose of supporting platform-level data models and platform functions in a platform management subsystem. PLDM is designed to be an effective interface and data model that provides efficient access to low-level platform inventory, monitoring, control, event, and data/parameters transfer functions. For example, temperature, voltage, or fan sensors can have a PLDM representation that can be used to monitor and control the platform using a set of PLDM messages. PLDM defines data representations and commands that abstract the platform management hardware.

Platform Level Data Model (PLDM) Base Specification

1 Scope

This specification describes base protocol elements of the Platform Level Data Model (PLDM) for the purpose of supporting platform-level data models and platform functions in a platform management subsystem. PLDM defines data representations and commands that abstract the platform management hardware.

This specification defines the following elements:

- the base Platform Level Data Model (PLDM) for various platform functions
- a common PLDM message format to support platform functions using PLDM

The PLDM message common fields support the identification of payload type, message, PLDM type, and PLDM command/completion codes.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 Approved References

DMTF DSP0004, *CIM Infrastructure Specification 2.5,* http://www.dmtf.org/standards/published_documents/DSP0004_2.5.0.pdf

DMTF DSP0241, *Platform Level Data Model (PLDM) over MCTP Binding Specification,* http://www.dmtf.org/standards/published_documents/DSP0241_1.0.0.pdf

DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes,* http://www.dmtf.org/standards/published_documents/DSP0245_1.0.0.pdf

IETF RFC4122*, A Universally Unique IDentifier (UUID) URN Namespace, July 2005* http://www.ietf.org/rfc/rfc4122.txt

ANSI/IEEE Standard 754, *Standard for Binary Floating Point Arithmetic*, http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=1316

2.2 Other References

Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba, *Advanced Configuration and Power Interface Specification 3.0*, ACPI, September 2, 2004, http://www.acpi.info/DOWNLOADS/ACPIspec30.zip

Intel, Hewlett-Packard, NEC, and Dell, *Intelligent Platform Management Interface Specification: Second Generation 2.0*, IPMI, 2004, ftp://download.intel.com/design/servers/ipmi/IPMIv2_0rev1_0markup.pdf

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards,* http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

OMG, *Unified Modeling Language (UML) from the Open Management Group (OMG),* http://www.uml.org/

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

3.1 Requirement Terms and Definitions

This clause defines key phrases and words that denote requirement levels in this specification.

3.1.1
can
used for statements of possibility and capability, whether material, physical, or causal

3.1.2
cannot
used for statements of possibility and capability, whether material, physical or causal

3.1.3
conditional
indicates requirements to be followed strictly to conform to the document when the specified conditions are met

3.1.4
mandatory
indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.1.5
may
indicates a course of action permissible within the limits of the document

3.1.6
need not
indicates a course of action permissible within the limits of the document

3.1.7
optional
indicates a course of action permissible within the limits of the document

3.1.8
shall
indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.1.9
shall not
indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.1.10

should

indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.1.11

should not

indicates that a certain possibility or course of action is deprecated but not prohibited

3.2 PLDM Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

3.1

baseboard management controller

BMC

a term coined by the IPMI specifications for the main management controller in an IPMI-based platform management subsystem. Also sometimes used as a generic name for a motherboard-resident management controller that provides motherboard-specific hardware monitoring and control functions for the platform management subsystem.

3.2

binary-coded decimal

BCD

indicates a particular binary encoding for decimal numbers where each four bits (*nibble*) in a binary number is used to represent a single decimal digit, and with the least significant four bits of the binary number corresponding to the least significant decimal digit

The binary values `0000b` through `1001b` represent decimal values 0 through 9, respectively. For example, with BCD encoding a byte can represent a two-digit decimal number where the most significant nibble (bits 7:4) of the byte contains the encoding for the most significant decimal digit and the least significant nibble (bits 3:0) contains the encoding for the least significant decimal digit (for example, 0010_1001b (0x29) in BCD encoding corresponds to the decimal number 29).

3.3

bridge

generically, the circuitry and logic that connect one computer bus or interconnect to another, allowing an agent on one to access the other

3.4

bus

a physical addressing domain shared between one or more platform components that share a common physical layer address space

3.5

byte

an 8-bit quantity. Also referred to as an *octet*.

NOTE: PLDM specifications shall use the term *byte*, not *octet*.

3.6

Common Information Model

CIM

the schema of the overall managed environment

It is divided into a *core, model*, *common model,* and *extended schemas*. For more information, see DSP0004.

3.7
endpoint
see *MCTP endpoint*

3.8
endpoint ID
EID
see *MCTP endpoint*

3.9
Globally Unique Identifier
GUID
see *UUID*

3.10
Inter-Integrated Circuit
I^{2}C
a multiple-master, two-wire, serial bus originally developed by Philips Semiconductor

3.11
idempotent command
a command that has the same effect for repeated applications of the same command

3.12
intelligent management device
IMD
a management device that is typically implemented using a microcontroller and accessed through a messaging protocol

Management parameter access provided by an IMD is typically accomplished using an abstracted interface and data model rather than through direct "register-level" access.

3.13
Intelligent Platform Management Interface
IPMI
a set of specifications defining interfaces and protocols originally developed for server platform management by the IPMI Promoters Group: Intel, Dell, HP, and NEC

3.14
Manageability Access Point
MAP
a collection of services of a system that provides management in accordance to CIM profiles and management protocol specifications published under the DMTF

3.15
managed entity
the physical or logical entity that is being managed through management parameters. Examples of *physical* entities include fans, processors, power supplies, circuit cards, chassis, and so on. Examples of *logical* entities include virtual processors, cooling domains, system security states, and so on.

3.16
Management Component Transport Protocol
MCTP
a media-independent transport protocol that was designed for intercommunication of low-level management messages within a platform management subsystem

3.17

management controller

a microcontroller or processor that aggregates management parameters from one or more management devices and makes access to those parameters available to local or remote software, or to other management controllers, through one or more management data models

Management controllers may also interpret and process management-related data, and initiate management-related actions on management devices. While a native data model is defined for PMCI, it is designed to be capable of supporting other data models, such as CIM, IPMI, and vendor-specific data models. The microcontroller or processor that serves as a management controller can also incorporate the functions of a management device.

3.18

management device

any physical device that provides protocol terminus for accessing one or more management parameters

A management device responds to management requests, but it does not initiate or aggregate management operations except in conjunction with a management controller (that is, it is a *satellite* device that is subsidiary to one or more management controllers). An example of a simple management device would be a temperature sensor chip. Another example would be a management controller that has I/O pins or built-in analog-to-digital converters that monitor state and voltages for a managed entity.

3.19

management parameter

a particular datum representing a characteristic, capability, status, or control point associated with a managed entity

Example management parameters include temperature, speed, volts, on/off, link state, uncorrectable error count, device power state, and so on.

3.20

MCTP bridge

an MCTP endpoint that can route MCTP messages (that are not destined for itself) that it receives on one interconnect to another without interpreting them

The ingress and egress media at the bridge may be either homogeneous or heterogeneous. Also referred to in this document as a "bridge".

3.21

MCTP bus owner

the entity that is responsible for MCTP EID assignment or translation on the buses of which it is a master

The MCTP bus owner may also be responsible for physical address assignment. For example, for SMBus bus segments, the MCTP bus owner is also the ARP master. This means the bus owner assigns dynamic SMBus addresses to devices that require it.

3.22

MCTP endpoint

a terminus or origin of an MCTP packet or message

The MCTP endpoint is identified by a value called the MCTP endpoint ID, or EID.

3.23

message

see *PLDM message*

3.24

message body

the portion of a PLDM message that carries the PLDM Type-specific data associated with the message

3.25

message originator

the original transmitter (source) of a message targeted to a particular PLDM terminus

3.26

most significant byte
MSB

the highest order byte in a number consisting of multiple bytes

3.27

non-idempotent command

a command that is not an idempotent command

3.28

nibble

the computer term for a four-bit aggregation, or half of a byte

3.29

payload

the information-bearing fields of a message

These fields are separate from the fields and elements (such as address fields, framing bits, checksums, and so on) that are used to transport the message from one point to another. In some instances, a given field may be both a payload field and a transport field.

3.30

physical transport binding

refers to specifications that define how a base messaging protocol is implemented on a particular physical transport type and medium, such as SMBus/I^{2}C, PCI Express™ Vendor Defined Messaging, and so on

3.31

Platform Level Data Model
PLDM

an internal-facing low-level data model that is designed to be an effective data/control source for mapping under the Common Information Model (CIM)

PLDM defines data structures and commands that abstract platform management subsystem components. PLDM supports a Type field to distinguish various types of messages and group them together based on the functions.

3.32

PLDM command

a command defined under the PLDM Type that is used for PLDM communications (for example, commands to control BIOS configuration and attributes transfer, perform SMBIOS data transfer, and monitor and control sensors)

3.33

PLDM message

a unit of communication based on the PLDM Type that is used for PLDM communications

3.34

PLDM message payload

a portion of the message body of a PLDM message

This portion of the message is separate from those fields and elements that are used to identify the payload type, message, PLDM Type, and PLDM command/completion codes.

3.35

PLDM request

Same as *PLDM command*. See 3.32.

3.36

PLDM request message

a message that is sent to a PLDM terminus to request a specific PLDM operation

A PLDM request message is acknowledged with a corresponding response message.

3.37

PLDM response

a response to a specific PLDM request

3.38

PLDM response message

a message that is sent in response to a specific PLDM request message

This message includes a "Completion Code" field that indicates whether the response completed normally.

3.39

PLDM terminus

identifies a set of resources within the recipient endpoint that is handling a particular PLDM message

3.40

Platform Management Component Intercommunications

PMCI

the name of a working group under the Distributed Management Task Force that is chartered to define standardized communication protocols, low-level data models, and transport definitions that support communications with and between management controllers and management devices that form a platform management subsystem within a managed computer system

3.41

point-to-point

refers to the case where only two physical communication devices are interconnected through a physical communication medium

The devices may be in a master and slave relationship, or the devices could be peers.

3.42

Universally Unique Identifier

UUID

an identifier originally standardized by the Open Software Foundation (OSF) as part of the Distributed Computing Environment (DCE). UUIDs are created using a set of algorithms that enables them to be independently generated by different parties without requiring that the parties coordinate to ensure that generated IDs do not overlap

In this specification, RFC4122 is used as the base specification for describing the format and generation of UUIDs. This identifier is also sometimes referred to as a globally unique identifier (GUID).

4 Symbols and Abbreviated Terms

The following symbols and abbreviations are used in this document.

4.1.
ACPI
Advanced Configuration and Power Interface

4.2.
ARP
Address Resolution Protocol

4.3.
CIM
Common Information Model

4.4.
DCE
Distributed Computing Environment

4.5.
GUID
Globally Unique Identifier

4.6.
IMD
intelligent management device

4.7.
IPMI
Intelligent Platform Management Interface

4.8.
ISO/IEC
International Organization for Standardization/International Engineering Consortium

4.9.
MC
Management Controller

4.10.
MCTP
Management Component Transport Protocol

4.11.
MSB
most significant byte

4.12.
OSF
Open Software Foundation

4.13.
PLDM
Platform Level Data Model

4.14.
PMCI
Platform Management Component Intercommunications

4.15.
TID
Terminus ID

4.16.
UUID
Universally Unique Identifier

4.17.
WBEM
Web-Based Enterprise Management

5 Conventions

The conventions described in the following clauses apply to all of the PLDM specifications.

5.1 Reserved and Unassigned Values

Unless otherwise specified, any reserved, unspecified, or unassigned values in enumerations or other numeric ranges are reserved for future definition by the DMTF.

Unless otherwise specified, numeric or bit fields that are designated as reserved shall be written as 0 (zero) and ignored when read.

5.2 Byte Ordering

Unless otherwise specified, for all PLDM specifications byte ordering of multi-byte numeric fields or multi-byte bit fields is "Little Endian" (that is, the lowest byte offset holds the least significant byte, and higher offsets hold the more significant bytes).

5.3 PLDM Data Types

Table 1 lists the abbreviations and descriptions for common data types that are used in PLDM message fields and data structure definitions.

Table 1 – PLDM Data Types

Data Type	Interpretation
uint8	Unsigned 8-bit binary integer
sint8	Signed 8-bit binary integer
uint16	Unsigned 16-bit binary integer
sint16	Signed 16-bit binary integer
uint32	Unsigned 32-bit binary integer
sint32	Signed 32-bit binary integer

Data Type	Interpretation
uint40	Unsigned 40-bit binary integer
sint40	Signed 40-bit binary integer
uint64	Unsigned 64-bit binary integer
sint64	Signed 64-bit binary integer
string	UCS-2 string
bool8	A Boolean value represented using an unsigned 8-bit binary integer where 0x00 means False, and any non-zero value means True
real32	Also known as "single precision". A 4-byte floating-point format, where: [31] – S (sign) bit (1 = negative, 0 = positive) [30:23] – exponent as a binary integer (8 bits) [22:0] – mantissa as a binary integer (23 bits) Per ANSI/IEEE Standard 754 convention, the value represented is determined as follows: If Exponent = 255 and Mantissa is nonzero, then Value = NaN ("Not a number"). If Exponent = 255 and Mantissa is zero and S is 1, then Value = -Infinity. If Exponent = 255 and Mantissa is zero and S is 0, then Value = Infinity. If 0<Exponent<255, then Value=(-1)**S * 2 ** (Exponent-127) * (1.Mantissa) where "1.Mantissa" is intended to represent the binary number created by prefixing Mantissa with an implicit leading 1 and a binary point. If Exponent = 0 and Mantissa is nonzero, then Value = (-1)**S * 2 ** (-126) * (0.Mantissa). These are "unnormalized" values. If Exponent = 0 and Mantissa is zero and S is 1, then Value = -0. * If Exponent = 0 and Mantissa is zero and S is 0, then Value = 0.
real64	Also known as "double-precison". A 8-byte floating-point, where: [63] – S (sign) bit (1 = negative, 0 = positive) [62:52] – exponent as a binary integer (11 bits) [51:0] – mantissa as a binary integer (52 bits) Per IEEE 754 convention, the value represented is determined as follows: If Exponent = 2047 and Mantissa is nonzero, then Value = NaN ("Not a number"). If Exponent = 2047 and Mantissa is zero and S is 1, then Value = -Infinity. If Exponent = 2047 and Mantissa is zero and S is 0, then Value = Infinity. If 0<Exponent<2047, then Value = (-1)**S * 2 ** (Exponent-1023) * (1.Mantissa) where "1.Mantissa" is intended to represent the binary number created by prefixing Mantissa with an implicit leading 1 and a binary point. If Exponent = 0 and Mantissa is nonzero, then Value = (-1)**S * 2 ** (-1022) * (0.Mantissa). These are "unnormalized" values. If Exponent = 0 and Mantissa is zero and S is 1, then Value = -0. * If Exponent = 0 and Mantissa is zero and S is 0, then Value = 0.
datetime	A string containing a date-time per DSP0004
char16	16-bit UCS-2 character
enum8	A sequential enumeration, starting from 0 as the default, with optional numeric declarator. The number 8 indicates that the enum is encoded using an 8-bit binary number. Example: enum8 { fred, mary, bob, george } has the value 0 correspond to fred, 1 for mary, 2 for bob, and 3 for george. A value may be explicitly declared such as: enum { fred, mary=2, bob, george }, in which case 0 corresponds to fred, 2 corresponds to mary, and 4 corresponds to george.

<table>
<tr><th>Data Type</th><th>Interpretation</th></tr>
<tr><td>timestamp104</td><td>A binary datetime type formatted as a series of 13 bytes, as follows:
(Generally, this format can be mapped to a CIM Datetime timestamp value.)
byte 12 UTC and Time resolution
The CIM Datetime format allows a variable number of significant digits to be represented for the date/time and UTC fields using a '*' character in the string to indicate which contiguous digit positions should be ignored, starting from the least significant position. PLDM generally supports this format by using this byte to present an enumeration for the resolution.
[7:4] UTC resolution = enum4 {UTCunspecified, minute, 10minute, hour }
[3:0] Time resolution = enum4 { microsecond, 10microsecond, 100microsecond, millisecond, 10millisecond, 100millisecond, second, 10second, minute, 10minute, hour, day, month, year }
bytes 11:10 year as uint16
byte 9 month as uint8 (starting with 1)
byte 8 day within the month as uint8 (starting with 1)
byte 7 hour within the day as uint8 (24-hour representation starting with 0)
byte 6 minute within the hour as uint8 (starting with 0)
byte 5 seconds within the minute as uint8 (starting with 0)
byte 4:2 microsecond within the second as a 24-bit binary integer (starting with 0)
bytes 1:0 UTC offset in minutes as sint16</td></tr>
<tr><td>interval72</td><td>A binary datetime interval formatted as a series of 9 bytes, as follows:
(Generally, this format can be mapped to a CIM Datetime interval value.)
byte 8 Time resolution
[7:4] reserved
[3:0] enum4 { microsecond, 10microsecond, 100microsecond, 1millisecond, 10millisecond, 100millisecond, second, 10second, minute, hour, day, 10day, 100day }
byte 7:6 number of days as uint16 (starting with 1)
NOTE: CIM DateTime specifies this as six-digit field.
byte 5 hour within the day as uint8 (24-hour representation starting with 0)
byte 4 minute within the hour as uint8 (starting with 0)
byte 3 seconds within the minute as uint8 (starting with 0)
bytes 2:0 microsecond within the second as a 24-bit binary integer (starting with 0)</td></tr>
<tr><td>ver32</td><td>A thirty-two-bit encoding of a version number. The encoding of the version number and alpha fields is defined in 5.5.
[31:24] = major version number
[23:16] = minor version number
[15:8] = update version number
[7:0] = "alpha" byte</td></tr>
<tr><td>UUID</td><td>See 5.4.</td></tr>
<tr><td>bitfield8</td><td>A byte with 8 bit fields. Each of these bit fields can be separately defined.</td></tr>
<tr><td>bitfield16</td><td>A 2-byte word with 16 bit fields. Each of these bit fields can be separately defined.</td></tr>
</table>

5.4 UUID

The format of the ID follows the byte (octet) format specified in RFC4122. RFC4122 specifies the following four different versions of UUID formats and generation algorithms suitable for use with PLDM:

- version 1 (0001b) ("time based")
- version 3 (0011b) "MD5 hash" ("name-based")
- version 4 (0100b) "Pseudo-random" ("name-based")
- version 5 "SHA1 hash" ("name-based")

The version 1 format is recommended. A UUID value should never change over the lifetime of the device or software version associated with the UUID.

For PLDM, the individual fields within the UUID are transferred in network byte order (most-significant byte first) per the convention described in RFC4122. For example, Table 2 shows byte order for a UUID in version 1 format.

Table 2 – Example UUID Format

Field	UUID Byte	MSB
time low	1	MSB
	2	
	3	
	4	
time mid	5	MSB
	6	
time high and version	7	MSB
	8	
clock seq high and reserved	9	
clock seq low	10	
Node	11	
	12	
	13	
	14	
	15	
	16	

5.5 Ver32 Encoding

The version field is comprised of four bytes referred to as the "major," "minor," "update," and "alpha" bytes. These bytes shall be encoded as follows:

- The "major," "minor," and "update" bytes are BCD-encoded, and each byte holds two BCD digits.
- The "alpha" byte holds an optional alphanumeric character extension that is encoded using the ISO/IEC 8859-1 Character Set.

- The semantics of these fields follow those in DSP4004.
- The value `0x00` in the alpha field means that the alpha field is not used. Software or utilities that display the version number should not display any characters for this field.
- The value `0xF` in the most-significant nibble of a BCD-encoded value indicates that the most-significant nibble should be ignored and the overall field treated as a single-digit value. Software or utilities that display the number should display only a single digit and should not put in a leading "0" when displaying the number.
- A value of `0xFF` in the "update" field indicates that the entire field is not present. `0xFF` is not allowed as a value for the "major" or "minor" fields. Software or utilities that display the version number should not display any characters for this field.

EXAMPLE:

Version 3.7.10a → `0xF3F71061`

Version 10.01.7 → `0x1001F700`

Version 3.1 → `0xF3F1FF00`

Version 1.0a → `0xF1F0FF61`

5.6 Notations

The following notations are used for PLDM specifications:

- M:N In field descriptions, this will typically be used to represent a range of byte offsets starting from byte M and continuing to and including byte N (M ≤ N). The lowest offset is on the left, and the highest is on the right.
- rsvd Abbreviation for Reserved. Case insensitive.
- [4] Square brackets around a number are typically used to indicate a bit offset. Bit offsets are given as zero-based values (that is, the least significant bit [LSb] offset = 0).
- [7:5] A range of bit offsets. The most-significant is on the left, and the least-significant is on the right.
- `1b` A lowercase "b" after a number consisting of `0`s and `1`s indicates that the number is in binary format.
- `0x12A` A leading "`0x`" indicates that the number is in hexadecimal format.

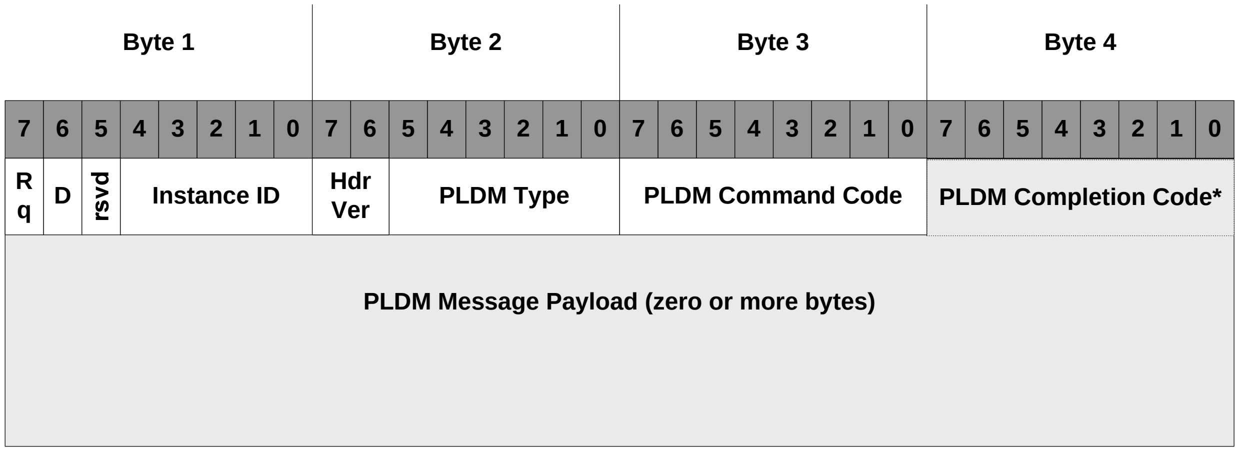
6 PLDM Base Protocol

The PLDM base protocol defines the common fields for PLDM messages and their usage.

Though there are command-specific PLDM header fields and trailer fields, the fields for the base protocol are common for all PLDM messages. These common fields support the identification of payload type, message, PLDM Type, and PLDM command/completion codes. The base protocol's common fields include a PLDM Type field that identifies the particular class of PLDM messages.

6.1 PLDM Message Fields

Figure 1 shows the fields that constitute a generic PLDM message. The fields within PLDM messages are transferred from the lowest offset first.



***The PLDM Completion Code is present only in PLDM response messages.**

Figure 1 – Generic PLDM Message Fields

Table 3 defines the common fields for PLDM messages.

Table 3 – PLDM Message Common Fields

Field Name	Field Size	Description
Rq	1 bit	Request bit. This bit is used to help differentiate between PLDM request messages and other PLDM messages. This field is set to 1b for PLDM request messages and unacknowledged datagram request messages. This field is set to 0b for PLDM response messages. See the following row of this table for valid combinations of Rq and D bits.
D	1 bit	Datagram bit. This bit is used to indicate whether the Instance ID field is being used for tracking and matching requests and responses, or just being used for asynchronous notifications. This field is set to 1b for asynchronous notifications. This field is set to 0b to indicate that the Instance ID field is being used for tracking and matching requests and responses. D and Rq bit combinations: 00b – For PLDM response messages 01b – For PLDM request messages 10b – Reserved 11b – For Unacknowledged PLDM request messages or asynchronous notifications
rsvd	1 bit	Reserved

Field Name	Field Size	Description
Instance ID	5 bits	The Instance ID (Instance Identifier) field is used to identify a new instance of a PLDM request to differentiate new PLDM requests that are sent to the same PLDM terminus. The Instance ID field is used to match up a particular instance of a PLDM response message with the corresponding instance of PLDM request message. If the requester issued a non-idempotent command, it shall complete any retries for that command before issuing a command with a new Instance ID.
Hdr Ver	2 bits	The Hdr Ver (Header Version) field identifies the header format. For this version of the specification, the value is set to 00b. This version applies to the PLDM message format.
PLDM Type	6 bits	The PLDM Type field identifies the type of PLDM that is being used in the control or data transfer carried out using this PLDM message. The PLDM Type field allows PLDM messages to be grouped together based on functions. See DSP0245 for the definitions of PLDM Type values.
PLDM Command Code	8 bits	For PLDM request messages, the PLDM Command Code field identifies the type of operation the message is requesting. The PLDM command code values are defined per PLDM Type. The PLDM Command Code that is sent in a PLDM request message shall be returned in the corresponding PLDM response message.
PLDM Message Payload	Variable	The PLDM message payload is zero or more bytes that are specific to a particular PLDM Message. By convention, the PLDM Message formats are described using tables with the first byte of the payload identified as byte 0. NOTE: The baseline PLDM message payload size is PLDM Type-specific.
PLDM Completion Code	8 bits	The PLDM Completion Code field provides the status of the operation. This field is the first byte of the PLDM Message Payload for PLDM response messages and is not present in PLDM request messages. This field indicates whether the PLDM command completed normally. If the command did not complete normally, then the completion code provides additional information regarding the error condition. The PLDM Completion Code can be generic or PLDM Type-specific.

6.2 Generic PLDM Completion Codes (PLDM_BASE_CODES)

The command completion code fields are used to return PLDM operation results in the PLDM response messages. On a successful completion of a PLDM operation, the specified response parameters (if any) shall also be returned in the response message. For a PLDM operation resulting in an error, unless otherwise specified, the responder shall not return any additional parametric data and the requester shall ignore any additional parameter data provided in the response.

Table 4 defines the generic completion codes for the PLDM commands. PLDM Type-specific command completion codes are defined in the respective PLDM specification. Unless otherwise specified in a PLDM specification, specific error completion codes are optional. If a PLDM command completes with an error, the generic failure message (ERROR), an appropriate generic error completion code from Table 4, or a PLDM Type-specific error completion code shall be returned. For an unsupported PLDM command, the ERROR_UNSUPPORTED_PLDM_CMD completion code shall be returned unless the responder is in a transient state (not ready), in which it cannot process the PLDM command. If the responder is in a transient state, it may return the ERROR_NOT_READY completion code.

Table 4 – Generic PLDM Completion Codes (PLDM_BASE_CODES)

Value	Name	Description
0x00	SUCCESS	The PLDM command was accepted and completed normally.
0x01	ERROR	This is a generic failure message to indicate an error processing the corresponding request message. It should not be used when a more specific error code applies.
0x02	ERROR_INVALID_DATA	The PLDM request message payload contained invalid data or an illegal parameter value.
0x03	ERROR_INVALID_LENGTH	The PLDM request message length was invalid. (The PLDM request message body was larger or smaller than expected for the particular PLDM command.)
0x04	ERROR_NOT_READY	The Receiver is in a transient state where it is not ready to process the corresponding PLDM command.
0x05	ERROR_UNSUPPORTED_PLDM_CMD	The command field in the PLDM request message is unspecified or not supported for this PLDM Type. This completion code shall be returned for any unsupported command values received.
0x20	ERROR_INVALID_PLDM_TYPE	The PLDM Type field value in the PLDM request message is invalid or unsupported.
0x80-0xFF	COMMAND_SPECIFIC	This range of completion code values is reserved for values that are specific to a particular PLDM request message. The particular values (if any) and their definition is provided in the specification for the particular PLDM command.
All other	Reserved	Reserved

6.3 Concurrent PLDM Command Processing

This section describes the specifications and requirements for handling concurrent overlapping PLDM requests.

6.3.1 Requirements for Responders

A PLDM terminus is not required to process more than one request at a time (that is, it can be "single threaded" and does not have to accept and act on new requests until it has finished responding to any previous request).

A responder that is not ready to accept a new request can either silently discard the request, or it can respond with an ERROR_NOT_READY message completion code.

The PLDM does not restrict any specific model for the number of requesters or responders that can communicate simultaneously. The PLDM specification allows an implementation to have a responder that handles one request at a time and to not maintain contexts for multiple requests or multiple requesters.

If a PLDM terminus is working on a request from a requester, then the PLDM terminus shall be able to process (or queue up processing) and send the response independently from sending its own request.

When a responder allows simultaneous communications with multiple requesters, the requirements on the responder are as follows:

- The responder shall use the following fields to track a PLDM request: the transport address (which is transport-binding specific, for example EID for MCTP transport) of the requester, PLDM Type, PLDM Command Code, and Instance ID of the PLDM request.
- If the responder runs out of internal resources, it may fail PLDM requests.

6.3.2 Requirements for Requesters

A PLDM terminus that issues PLDM requests to another PLDM terminus shall wait until it either gets the response to a particular request, times out waiting for the response, or receives an indication that transmission of the particular request failed, before issuing a new PLDM request.

A PLDM terminus that issues PLDM requests is allowed to have multiple simultaneous requests outstanding to *different* responders.

A PLDM terminus that issues PLDM requests should be prepared to handle the order of responses that may not match the order in which the requests were sent (that is, it should not automatically assume that a response that it receives is in the order in which the request was sent). It should check to see that the PLDM Type, PLDM Command Code, and Instance ID values in the response match up with a corresponding outstanding command before acting on any parameters returned in the response.

The timing specifications shown in Table 5 are specific to PLDM request messages. The PLDM responses are not retried. A “try” or “retry” of a request is defined as a complete transmission of the PLDM request message.

Table 5 – Timing Specifications for PLDM Messages

Timing Specification	Symbol	Min	Max	Description
Number of request retries	PN1	2	See "Description"	Total of three tries, minimum: the original try plus two retries. The maximum number of retries for a given request is limited by the requirment that all retries shall occur within $PT3_{Max}$ of the initial request.
Request-to-response time	PT1	–	100 msec	This interval is measured at the responder from the end of the reception of the PLDM request to the beginning of the transmission of the response. This requirement is tested under the condition where the responder can successfully transmit the response on the first try.
Time-out waiting for a response	PT2	$PT1_{Max}$+ $2*PT4_{Max}$	$PT3_{Min}$ – $2*PT4_{Max}$	This interval is measured at the requester from the end of the successful transmission of the PLDM request to the beginning of the reception of the corresponding PLDM response. This interval at the requester sets the minimum amount of time that a requester should wait before retrying a PLDM request. Note: This specification does not preclude an implementation from adjusting the minimum time-out waiting for a response to a smaller number than PT2 based on measured response times from responders. The mechanism for doing so is outside the scope of this specification.

Timing Specification	Symbol	Min	Max	Description
Instance ID expiration interval	PT3	5 sec [1]	6 sec	This is the interval after which the Instance ID for a given response will expire and become reusable if a response has not been received for the request. This is also the maximum time that a responder tracks an Instance ID for a given request from a given requester.
Transmission Delay	PT4	–	100 ms	Time to take into account transmission delay of a PLDM Message. Measured as the time between the end of the transmission of a PLDM message at the transmitter to the beginning of the reception of the PLDM message at the receiver.

NOTE: [1] If a requester is reset, it may produce the same Instance ID for a request as one that was previously issued. To guard against this, it is recommended that Instance ID expiration be implemented. Any request from a given requester that is received more than PT3 seconds after a previous, matching request should be treated as a new request, not a retry.

7 PLDM Messaging Control and Discovery Commands

The PLDM base definition supports a PLDM Type field that allows the commands to be grouped using a PLDM Type. This section contains detailed descriptions for PLDM messages that are used for control and discovery operations. The PLDM commands for PLDM messaging control and discovery are also defined in this section.

Table 6 defines the PLDM command codes for PLDM messaging control and discovery.

Table 6 – PLDM Messaging Control and Discovery Command Codes

Command	Code Value	Requirement	Section
SetTID	0x01	Optional	See 7.1.1.
GetTID	0x02	Mandatory	See 7.1.2.
GetPLDMVersion	0x03	Mandatory	See 7.2.
GetPLDMTypes	0x04	Mandatory	See 7.3.
GetPLDMCommands	0x05	Mandatory	See 7.4.

7.1 PLDM Terminus

A PLDM Terminus is defined as the point of communication termination for PLDM messages and the PLDM functions associated with those messages. Given a PLDM terminus, a mechanism is required that can uniquely identify each terminus so that the semantic information can be bound to that identification. The Terminus ID (TID) is a value that identifies a PLDM terminus. TIDs are used in PLDM messages when it is necessary to identify the PLDM terminus that is the source of the PLDM Message. TIDs are defined within the scope of PLDM Messaging.

7.1.1 SetTID Command

The SetTID command is used to set the Terminus ID (TID) for a PLDM Terminus. This command is typically only used by the PLDM Initialization Agent function. The command format is shown in Table 7.

Table 7 – SetTID Command Format

Byte	Type	Request Data
0	uint8	**TID** Special value: 0x00, 0xFF = reserved.
Byte	**Type**	**Response Data**
0	enum8	**completionCode** Possible values: { PLDM_BASE_CODES }

7.1.2 GetTID Command

The GetTID command is used to retrieve the present Terminus ID (TID) setting for a PLDM Terminus. The command format is shown in Table 8.

Table 8 – GetTID Command Format

Byte	Type	Request Data
0	–	No request data
Byte	**Type**	**Response Data**
0	enum8	**completionCode** possible value: { PLDM_BASE_CODES }
1	uint8	**TID** special value: 0x00 – Unassigned TID, 0xFF – reserved

7.2 GetPLDMVersion

The GetPLDMVersion command can be used to retrieve the PLDM base specification versions that the PLDM terminus supports, as well as the PLDM Type specification versions supported for each PLDM Type. The format of the request and response message parameters for this command is shown in Table 9.

More than one version number can be returned for a given PLDM Type by the GetPLDMVersion command. This enables the command to be used for reporting different levels of compatibility and for backward compatibility with different specification versions. The individual specifications for the given PLDM Type define the requirements for which version number values should be used for that PLDM Type. Those documents define which earlier version numbers, if any, shall also be listed.

The command returns a completion code that indicates whether the PLDM Type number passed in the request is supported. This enables the command to also be used to query the endpoint for whether it supports a given PLDM Type.

Table 9 – GetPLDMVersion Request and Response Message Format

Byte	Type	Request Data
0:3	uint32	**DataTransferHandle** This field is a handle that is used to identify PLDM version data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart.
4	enum8	**TransferOperationFlag** This field is an operation flag that indicates whether this is the start of the transfer. Value: {GetNextPart=0x00, GetFirstPart=0x01}
5	uint8	**PLDMType** This field identifies the PLDM Type whose version information is being requested. See DSP0245 for valid PLDMType values.
Byte	**Type**	**Response Data**

0	enum8	**CompletionCode**
		possible values:
		{
		PLDM_BASE_CODES,
		INVALID_DATA_TRANSFER_HANDLE=0x80,
		INVALID_TRANSFER_OPERATION_FLAG=0x81,
		INVALID_PLDM_TYPE_IN_REQUEST_DATA=0x83
		}

1:4	uint32	**NextDataTransferHandle** This field is a handle that is used to identify the next portion of PLDM version data transfer.
5	enum8	**TransferFlag** This field is the transfer flag that indicates what part of the transfer this response represents. Possible values: {Start=0x01, Middle=0x02, End=0x04, StartAndEnd = 0x05}
Variable	–	**Portion of PLDMVersionData** (contains one or more version fields as described in Table 10) See Table 10 for the format.

Table 10 – PLDM Representation of PLDMVersionData

Byte	Type	Field
0:3	ver32	**Version[0]** This field is the first entry of the version supported for the specified PLDM type.
…	…	…
4*(N-1):4*N-1	ver32	**Version[N-1]** This field is the Nth entry of the version supported for the specified PLDM type.
4*N:4*N+3	uint32	**PLDMVersionDataIntegrityChecksum** Integrity checksum on the PLDM version data. It is calculated starting at the first byte of the PLDM representation of PLDMVersionData. For this specification, CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$ (same as the one used by IEEE 802.3) shall be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first.

This command is defined in such a manner that it allows the PLDM version data to be transferred using a sequence of one or more command or response messages. When more than one command is used to transfer the PLDM version data, the response messages contain the non-overlapping contiguous portions of PLDM version data as defined in Table 10. By combining the portions of PLDM version data from the response messages, the entire PLDM version data can be reconstructed.

The version of this PLDM base specification shall be 1.0.0 (major version number 1, minor version number 0, update version number 0, and no alpha version).

This is reported using the encoding as: `0xF1F0F000.`

7.3 GetPLDMTypes

The GetPLDMTypes command enables management controllers to discover the PLDM type capabilities supported by the PLDM terminus and get a list of the PLDM types that are supported. The request and response parameters for this message are listed in Table 11.

The response to this command may be specific according to which transport endpoint over which the request was received (that is, a device that supports a given PLDM Type on a transport endpoint may not support that PLDM Type equally across all the transport endpoints that connect to the device).

Table 11 – GetPLDMTypes Request and Response Message Format

Byte	Type	Request Data
-	-	None
Byte	**Type**	**Response Data**
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES}
1:8	bitfield8[8]	**PLDMTypes** Each bit represents whether a given PLDM Type is supported: 1b = PLDM Type is supported. 0b = PLDM Type is not supported. For bitfield8[N], where N = 0 to 7 [7] – PLDM Type N*8+7 Supported [..] – … [1] – PLDM Type N*8+1 Supported [0] – PLDM Type N*8+0 Supported

7.4 GetPLDMCommands

The GetPLDMCommands command enables management controllers to discover the PLDM command capabilities supported by the PLDM terminus for a specific PLDM Type and version as a responder. The request and response parameters for this message are listed in Table 12.

The response to this command may be specific according to which transport endpoint over which the request was received (that is, a device that supports a given PLDM Type on a transport endpoint may not support that PLDM Type equally across all the transport endpoints that connect to the device).

Table 12 – GetPLDMCommands Request and Response Message Format

Byte	Type	Request Data
0	uint8	**PLDMType** This field identifies the PLDM Type for which command support information is being requested. See DSP0245 for valid PLDMType values.
1:4	ver32	**Version** This field identifies the version for the specified PLDM Type.
Byte	**Type**	**Response Data**

0	enum8	**CompletionCode**
		Possible values:
		{
		PLDM_BASE_CODES,
		INVALID_PLDM_TYPE_IN_REQUEST_DATA=0x83
		INVALID_PLDM_VERSION_IN_REQUEST_DATA=0x84
		}

1:32	bitfield8[32]	**PLDMCommands** (up to 256 commands supported for the specified PLDM Type) Each bit represents whether a given PLDM command is supported: 1b = PLDM command is supported. 0b = PLDM command is not supported. For bitfield8[N], where N = 0 to 31 [7] – PLDM Command N*8+7 Supported [..] – … [1] – PLDM Command N*8+1 Supported [0] – PLDM Command N*8 Supported

8 PLDM Messaging Control and Discovery Examples

The GetPLDMVersion command (see 7.2) for transferring PLDM version data supports multipart transfers. The GetPLDMVersion command uses flags and data transfer handles to perform multipart transfers. The following requirements apply to the usage of TransferOperationFlag, TransferFlag, and DataTransferHandle for a given data transfer:

1) For initiating a data transfer (or getting the first part of data) by using a Get* command, the TransferOperationFlag shall be set to GetFirstPart in the request of the Get* command.

   - For transferring any part of the data other than the first part by using a Get* command, the TransferOperationFlag shall be set to GetNextPart and the DataTransferHandle shall be set to the NextDataTransferHandle that was obtained in the response of the previous Get* command for this data transfer.
   - The TransferFlag specified in the response of a Get* command has the following meanings:
     – Start, which is the first part of the data transfer.

  - Middle, which is neither the first nor the last part of the data transfer.
  - End, which is the last part of the data transfer.
  - StartAndEnd, which is the first and the last part of the data transfer.
- The requester shall consider a data transfer complete when the TransferFlag in the response of a Get* command is set to End or StartAndEnd.

EXAMPLE 1: The example in Figure 2 shows how multipart transfers can be performed using the generic mechanism defined in the GetPLDMVersion command. In Figure 2, the PLDM version data is transferred in three parts. Figure 2 shows the flow of the data transfer.

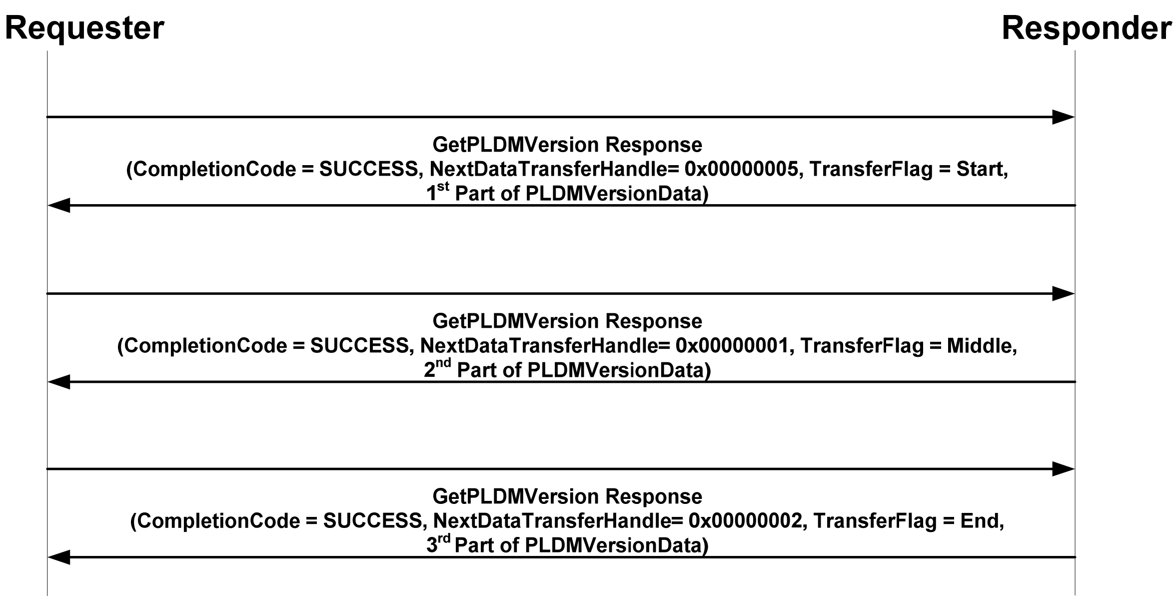


Figure 2 – Example of Multipart PLDM Version Data Transfer Using the GetPLDMVersion Command

EXAMPLE 2: Figure 3 shows an example sequence of steps performed by a requester to discover the PLDM versions and types supported by the responder as well as the commands supported for each PLDM type.

In the example, the following steps are performed by the requester:

1) The requester first uses the GetTID command to get the PLDM Terminus ID of the responder.

2) The requester then uses GetPLDMTypes to discover the PLDM types supported by the responder. (In the example shown in Figure 3, the responder supports two PLDM types, PLDM Type 0 and PLDM Type 1.)

3) For each PLDM type that is supported by the responder, the requester uses GetPLDMVersion and GetPLDMCommands to discover the supported versions of the specifications for the PLDM type and the supported PLDM commands for the specific PLDM version and type. In this example, the responder supports only one version of the specification (1.0.0) for each PLDM Type.

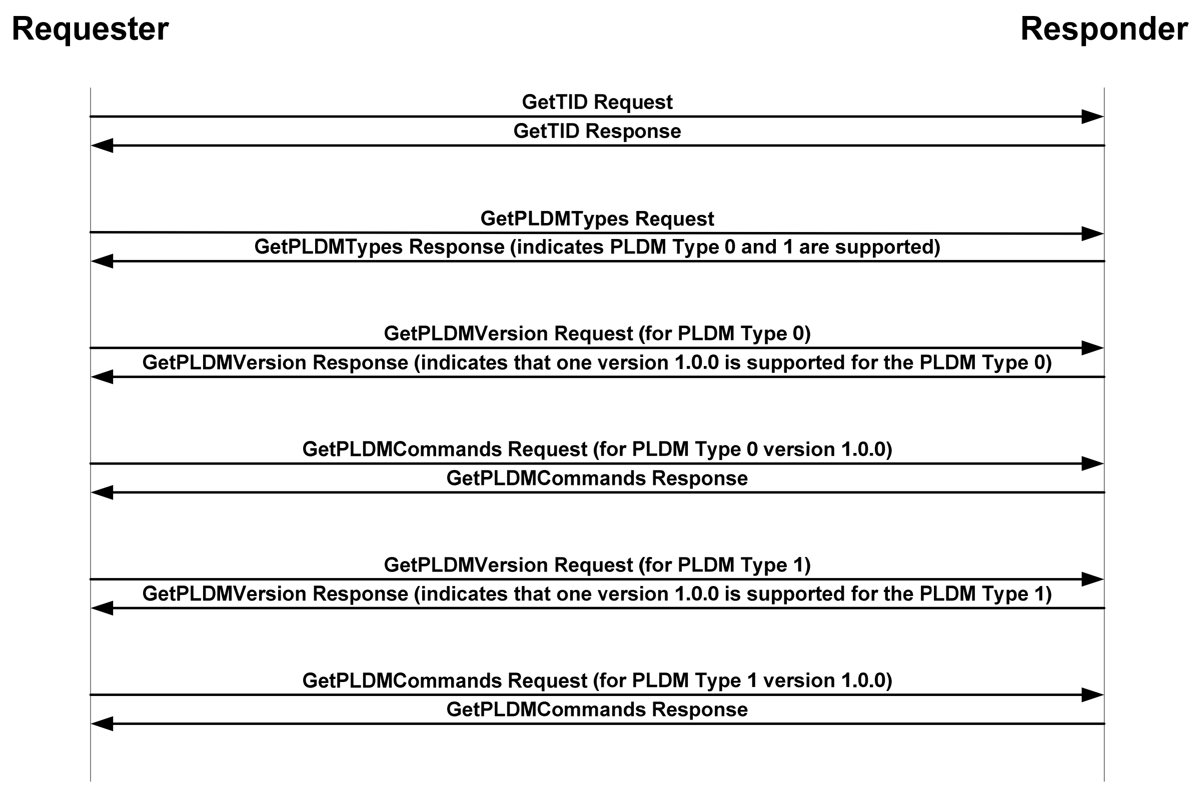


Figure 3 – PLDM Discovery Command Example

ANNEX A
(Informative)

Change Log

Version	Date	Author	Description
1.0.0a	9/24/2008	Hemal Shah	1.0.0a Preliminary version
1.0.0	4/23/2009		DMTF Standard Release