# NVM Express

# Management Interface

**Revision 1.0**

**November 17, 2015**

Please send comments and questions to *info@nvmexpress.org*

NVM Express Management Interface revision 1.0 specification available for download at http://nvmexpress.org. NVM Express Management Interface revision 1.0 ratified on November 17th, 2015.

**LEGAL NOTICE:**

**LEGAL DISCLAIMER**:

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "**AS IS**" BASIS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, NVM EXPRESS, INC. (ALONG WITH THE CONTRIBUTORS TO THIS DOCUMENT) HEREBY DISCLAIM ALL REPRESENTATIONS, WARRANTIES AND/OR COVENANTS, EITHER EXPRESS OR IMPLIED, STATUTORY OR AT COMMON LAW, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, VALIDITY, AND/OR NONINFRINGEMENT.

NVM Express Workgroup
c/o Virtual, Inc.
401 Edgewater Place, Suite 600
Wakefield, MA 01880
info@nvmexpress.org

Peter Z. Onufryk

Workgroup Chair

John W. Carroll

Specification Editor

**Table of Contents**

# 1 Introduction

## 1.1 Overview

NVM Express (NVMe) is a register-level interface that allows in-band host software to communicate with an NVM Subsystem. The NVMe Management Interface (NVMe-MI) allows a Management Controller to communicate out-of-band with an NVMe NVM Subsystem over one or more external interfaces.

Since this specification builds on the NVMe specification, knowledge of NVMe is assumed.

## 1.2 Scope

This specification defines an architecture and command set for out-of-band management of an NVMe NVM Subsystem.

NVMe-MI has the following key capabilities:

- Discover devices that are present and learn capabilities of each device
- Store data about the host environment enabling a Management Controller to query the data later
- Health and temperature monitoring
- Multiple Command Slots to prevent a long latency command from blocking monitoring operations
- Processor and operating system agnostic
- A standard format for VPD and defined mechanisms to read/write VPD contents
- Preserves data at rest security

### 1.2.1 Outside of Scope

The architecture and command set are specified apart from any usage model. This specification does not specify whether NVMe is used to implement a solid-state drive, a main memory, a cache memory, a backup memory, a redundant memory, etc. Specific usage models are outside the scope, optional, and not licensed.

This interface is NVM technology agnostic and is specified at a level that abstracts implementation details associated with any specific NVM technology. For example, NAND wear leveling, block erases, and other management tasks are abstracted.

The implementation or use of other published specifications referred to in this specification, even if required for compliance with the specification, are outside the scope of this specification (e.g., PCI Express, SMBus/I2C and MCTP).

The management of NVMe FRUs containing multiple architecturally visible NVM subsystems is outside the scope of this specification.This specification does not define new security mechanisms.

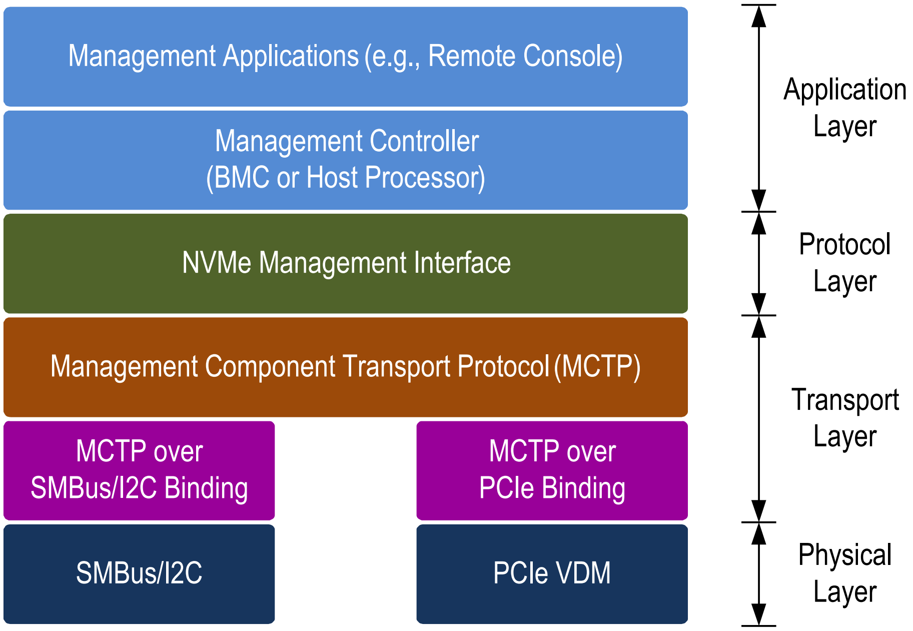
This specification does not cover management of non-transparent bridges, PCIe switches or management using any interface other than MCTP over PCIe VDM or SMBus/I2C. Co-ordination between multiple Management Controllers or a Management Controller and a device other than a Management Endpoint is outside the scope of this specification.

Coordinating concurrency resulting from operations associated with multiple Management Endpoints or between a host and Management Endpoint operations is outside the scope of this specification.

## 1.3 Theory of Operation

NVMe-MI is designed to provide a common interface over multiple physical layers (i.e., PCI Express, SMBus/I2C) for inventory, monitoring, configuration, and change management. The interface provides the flexibility necessary to manage NVM Subsystems using an out-of-band mechanism in a variety of host environments and systems.

**Figure 1: NVMe Management Interface Protocol Layering**


Management Applications (e.g., Remote Console)
Management Controller (BMC or Host Processor)
NVMe Management Interface
Management Component Transport Protocol (MCTP)
MCTP over SMBus/I2C Binding
MCTP over PCIe Binding
SMBus/I2C
PCIe VDM
Application Layer
Protocol Layer
Transport Layer
Physical Layer

NVMe-MI utilizes the Management Component Transport Protocol (MCTP) as the command transport and utilizes existing MCTP SMBus/I2C and PCIe bindings for the physical layer. MCTP commands are submitted to one of two Command Slots associated with each Management Endpoint.

### 1.4 Architectural Model

An NVMe storage device, such as a PCIe SSD, that implements this specification, consists of an NVMe NVM Subsystem with one or more PCIe ports and an optional SMBus/I2C port. Each port has a Port Identifier that is less than or equal to the Number of Ports (NUMP) field value in the NVM Subsystem Information Data Structure. The port identifier for a PCIe port is the same as the Port Number field in the PCIe Link Capabilities Register.

NVMe-MI supports Vital Product Data (VPD) that utilizes the format defined in the IPMI Platform Management FRU Information Storage Definition and is stored in a FRU Information Device. The FRU Information Device may be implemented in the NVM Subsystem, in an external device (e.g., serial EEPROM), or a combination of the two. The VPD is accessible over any port that supports NVMe-MI using MCTP commands. If the NVMe storage device has an SMBus/I2C interface, then the VPD is accessible using the access mechanism over I2C as defined in the IPMI Platform Management FRU Information Storage Definition.

Figure 2 illustrates a single-port PCIe SSD with the FRU Information Device implemented by the NVM Subsystem. Figure 3 illustrates a dual-port PCIe SSD with an SMBus/I2C port and a FRU Information Device implemented using a Serial EEPROM.

**Figure 2: Single-Port PCIe SSD**

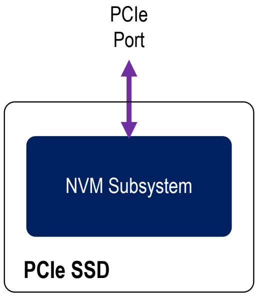
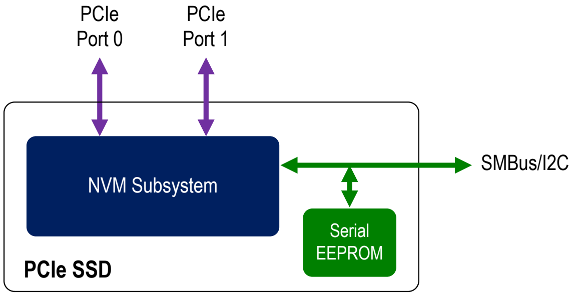


**Figure 3: Dual-Port PCIe SSD with SMBus/I2C**



The NVMe Management Interface is used to send Command Messages which consist of standard NVMe Admin Commands that target a Controller within the NVM Subsystem; commands that provide access to the PCI Express configuration, I/O, and memory spaces of a Controller in the NVM Subsystem; and Management Interface specific commands for inventorying, configuring and monitoring of the NVM Subsystem. Each Management Endpoints advertises its unique capabilities. All Management Endpoints may support the same commands even though PCIe ports are full duplex with much higher data rates than SMBus (i.e., both SMBus/I2C and PCIe VDM are capable of providing the same functionality).

The PCIe ports and SMBus/I2C port of an NVM Subsystem may optionally each contain a single NVMe Management Endpoint (hereafter referred to as simply Management Endpoint). A Management Endpoint is an MCTP endpoint that is the terminus and origin of MCTP packets/messages and is responsible for implementing the MCTP Base Protocol, processing MCTP Control Messages, and internal routing of Command Messages.

Each NVMe Controller in the NVM Subsystem shall provide an NVMe Controller Management Interface (hereafter referred to as simply Controller Management Interface). The Controller Management Interface executes Controller operations on behalf of any Management Endpoint in the NVM Subsystem. Management Endpoints may route commands to any NVMe Controller in the NVM Subsystem. A Controller Management Interface logically executes one operation at a time. A Controller Management Interface is not precluded from executing two or more operations in parallel; however, there shall always be an equivalent pattern of sequential operations with the same results.

Figure 4 illustrates an example NVM Subsystem corresponding to the PCIe SSD shown in Figure 2. The NVM Subsystem contains a single Controller and there is a Management Endpoint associated with the PCIe port.

Figure 4: NVM Subsystem Associated with Single Ported PCIe SSD

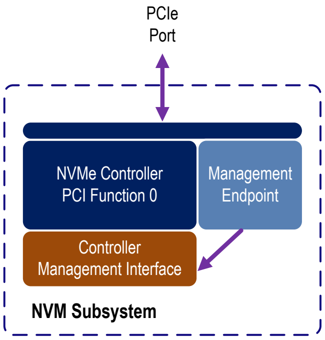
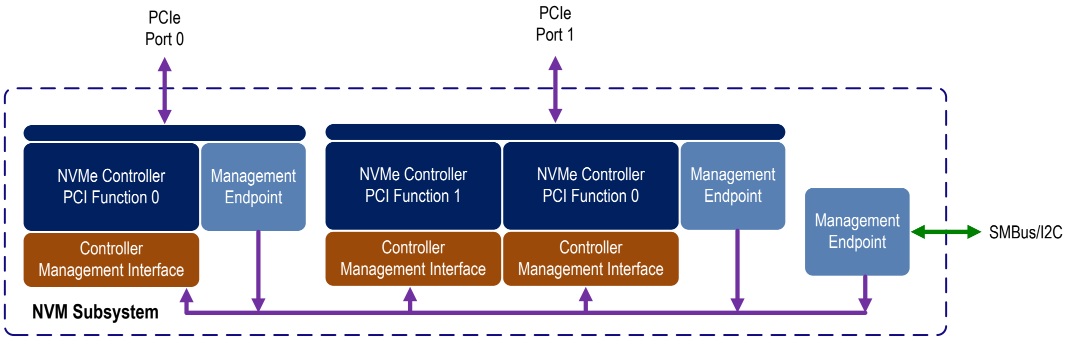


Figure 5 illustrates an example NVM Subsystem corresponding to the PCIe SSD shown in Figure 3. The NVM Subsystem contains one Controller associated with PCIe Port 0 and two Controllers associated with PCIe Port 1. There is a Management Endpoint associated with the each PCIe port and the SMBus/I2C port. Since the NVM Subsystem contains a Management Endpoint, all Controllers have an associated Controller Management Interface.

Figure 5: NVM Subsystem Associated with Dual Ported PCIe SSD with SMBus/I2C



Management Interface Request Messages and Response Messages are transported as MCTP messages with the Message Type set to NVM Express Management Messages over MCTP (refer to the MCTP IDs and Codes specification). All Command Messages originate with the Management Controller and result in a Response Message from the Management Endpoint.

## 1.5 Conventions

Hardware shall return zero for all bits, fields, and registers that are marked as reserved. The Management Controller should not rely on a value of zero being returned as future revisions of this specification may contain non-zero values. The Management Controller should write all reserved bits and registers with the value of zero. Future revisions of this specification may rely on a zero value being written for backward compatibility.

Some fields or registers are 0's based values.  In a 0's based value, the value of 0h corresponds to 1; other values similarly correspond to the value+1.

### 1.5.1 Definitions

#### 1.5.1.1 Controller or NVMe Controller

Refer to the NVM Express specification

#### 1.5.1.2 Controller Management Interface or NVMe Controller Management Interface

An interface associated with each NVMe Controller in the NVM Subsystem that is responsible for executing management operations on behalf of a Management Endpoint.

#### 1.5.1.3 Management Controller

A device (e.g., BMC) responsible for platform management that uses the NVM Express Management Interface to communicate to Management Endpoints.

#### 1.5.1.4 Management Endpoint or NVMe Management Endpoint

An MCTP endpoint associated with an NVM Subsystem (e.g., an NVMe SSD) that is the terminus and origin of MCTP packets/messages and which processes Request Messages.

#### 1.5.1.5 VPD or Vital Product Data

Field Replaceable Unit (FRU) Information which may be stored in a FRU Information Device.

#### 1.5.1.6 FRU Information Device

A storage device (e.g., serial EEPROM) used to store Vital Product Data.

#### 1.5.1.7 Command Slot

A logical target within a Management Endpoint where a Management Controller sends a Request Message. Each Management Endpoint has exactly two Command Slots.

#### 1.5.1.8 Request Message

An MCTP message originating from a Management Controller. A Request Message may be a Command Message, a Control Primitive, or another type of MCTP message.

#### 1.5.1.9 Command Message

A type of Request Message that contains an NVMe Admin Command, PCIe Command, or NVMe-MI Command.

#### 1.5.1.10 Control Primitive

A type of Request Message that may be sent while a Command Slot is processing a Command Message. A single packet MCTP message used to convey an NVMe-MI control request.

#### 1.5.1.11 Response Message

An MCTP message originating from a Management Endpoint in response to a Request Message.

#### 1.5.1.12 NVM Subsystem

Refer to the NVM Express specification.

### 1.5.2 Keywords

Several keywords are used to differentiate between different levels of requirements.

#### 1.5.2.1 mandatory

A keyword indicating items to be implemented as defined by this specification.

#### 1.5.2.2 may

A keyword that indicates flexibility of choice with no implied preference.

#### 1.5.2.3 optional

A keyword that describes features that are not required by this specification. However, if any optional feature defined by the specification is implemented, the feature shall be implemented in the way defined by the specification.

#### 1.5.2.4 R

“R” is used as an abbreviation for “reserved” when the figure or table does not provide sufficient space for the full word “reserved”.

#### 1.5.2.5 reserved

A keyword indicating reserved bits, bytes, words, fields, and opcode values that are set-aside for future standardization. Their use and interpretation may be specified by future extensions to this or other specifications. A reserved bit, byte, word, field, or register shall be cleared to zero, or in accordance with a future extension to this specification. The recipient shall not check the value of reserved bits, bytes, words, or fields.

#### 1.5.2.6 shall

A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to the specification.

#### 1.5.2.7 should

A keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “it is recommended”.

## 1.6 Conventions

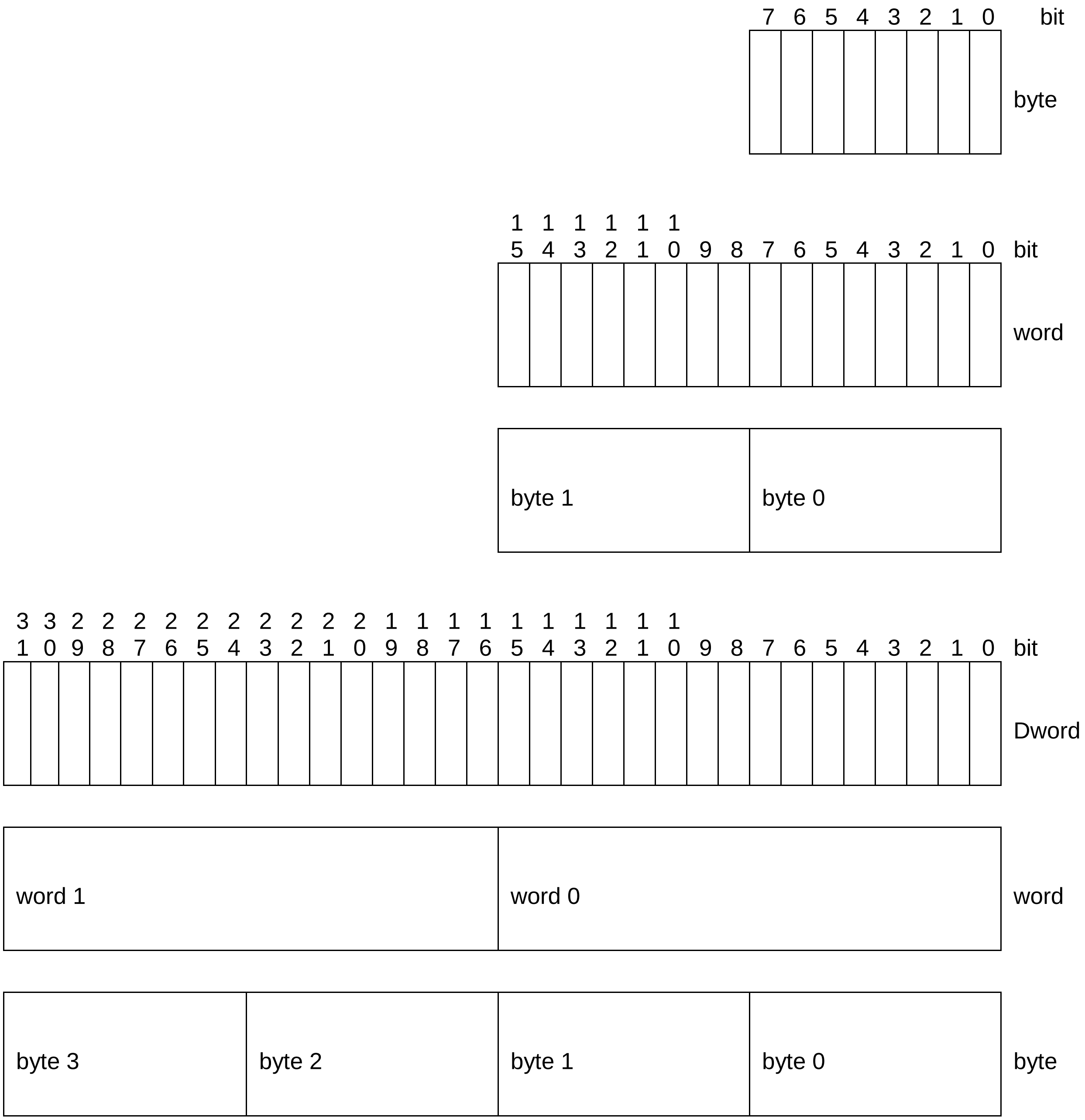
A 0-based value is a numbering scheme for which the number 0h actually corresponds to a value of 1h and thus produces the pattern of 0h = 1h, 1h = 2h, 2h = 3h, etc. In this numbering scheme, there is not a method for specifying the value of 0h.

Some parameters are defined as a string of ASCII or UTF-8 characters. ASCII data fields shall contain only code values 20h through 7Eh. UTF-8 is backwards compatible with ASCII encoding and supports additional characters with variable length encoding. For the string “Copyright”, the character “C” is the first byte, the character “o” is the second byte, etc. The string is left justified and shall be padded with spaces (ASCII character 20h) to the right if necessary.

### 1.6.1 Byte, Word and Dword Relationships

Figure 6 illustrates the relationship between bytes, words and Dwords. This specification specifies data in a little endian format.

**Figure 6: Byte, word and Dword Relationships**



## 1.7 References

I2C Bus specification, revision 6.0. http://www.i2c.org

IPMI Platform Management FRU Information Storage Definition 1.0, Version 1.2. Available from http://www.intel.com

MCTP Base Specification (DSP0236), version 1.2.1. Available from http://www.dmtf.org.

MCTP IDs and Codes (DSP0239), version 1.3.0. Available from http://dmtf.org.

MCTP PCIe VDM Transport Binding Specification (DSP0238), version 1.0.2. Available from http://www.dmtf.org.

MCTP SMBus/I2C Transport Binding Specification (DSP0237), version 1.0.0. Available from http://www.dmtf.org.

NVM Express specification, revision 1.2.  Available from http://www.nvmexpress.org

NVMe™ (NVM Express™) Management Messages over MCTP Binding specification (DSP0235), revision 1.0.0. Available from http://www.dmtf.org.

PCI specification, revision 3.0. Available from http://www.pcisig.com.

PCI Express specification, revision 3.1. Available from http://www.pcisig.com.

System Management Bus (SMBus) Specification, revision 3.0. Available from http://www.smbus.org.

# 2 Physical Layer

This section describes the physical layers supported by NVMe-MI. An implementation may support zero or more PCIe ports and an optional SMBus/I2C port. An implementation shall support at least one port.

## 2.1 PCI Express

A PCIe port in an NVM subsystem may implement a Management Endpoint. If the PCIe port implements a Management Endpoint, the PCIe port shall support MCTP over PCIe Vendor Defined Messages (VDMs) as specified by the Management Component Transport Protocol (MCTP) PCIe VDM Transport Binding Specification.

## 2.2 SMBus/I2C

If the NVM Subsystem implements an SMBus/I2C interface and associated with that SMBus/I2C interface is a Management Endpoint, then the interface shall support MCTP over SMBus/I2C as specified by the Management Component Transport Protocol (MCTP) SMBus/I2C Transport Binding Specification.

If the NVM Subsystem implements an SMBus/I2C interface, then the NVM Subsystem may optionally support the NVMe Basic Management Command for health and status polling. The NVMe Basic Management Command is defined in Appendix A – Technical Note: NVM Express Basic Management Command. It is possible to support both MCTP and the Basic Management Command.

The SMBus/I2C Management Endpoint shall be accessible at a power-up SMBus/I2C address of 0x3A and should be SMBus ARP-capable (as defined in the SMBus 3.0 specification).[1] If the NVM Subsystem is “Discoverable” (as defined in the SMBus 3.0 specification), the device shall issue a “Notify ARP Master” command when the NVM Subsystem is ready to communicate.

If the NVM Subsystem implements an SMBus/I2C interface, then VPD information shall be accessible from the Management Endpoint using Sequential Read and Random Read operations as defined by the IPMI Platform Management FRU Information Storage Definition specification.

The VPD shall be accessible using I2C read operations from a FRU Information Device at a power-up SMBus/I2C address of 0xA6 and should be SMBus ARP-capable (as defined in the SMBus 3.0 specification).[2] If the FRU Information Device is “Discoverable” (as defined in the SMBus 3.0 specification), it shall issue a “Notify ARP Master” command when the FRU Information Device is ready to communicate.

If ARP is supported, then the SMBus/I2C Management Endpoint and VPD shall both use the SMBus Address Resolution Protocol Unique Device Identifier (UDID) shown in Figure 7. The only difference between the NVM Subsystem and FRU Information Device UDID is the most significant bit of the Vendor Specific ID. This fact may be used by the MCTP bus owner to associate an SMBus/I2C Management Endpoint with its corresponding VPD.

Clock stretching is allowed by the Management Controller, Management Endpoint, and the VPD. However, implementations are strongly discouraged from using clock stretching so that communications are more predictable with higher throughput.

When a NACK is received, a Management Endpoint shall follow the MCTP specification for a non-bridge endpoint. The Management Endpoint treats a STOP condition due to excessive SMBus NACKs as an

[1] The address 3Ah appears on SMBus as 0b0011_101x where x represents the SMBus read/write bit.
[2] The address A6h appears on SMBus as 0b1010_011x where x represents the SMBus read/write bit.

implicit Pause Control Primitive. Refer to 4.4.

**Figure 7: NVM Subsystem and FRU Information Device SMBus UDID**

<table>
<tr><th>Bits</th><th>Field</th><th colspan="2">Description</th></tr>
<tr><td rowspan="5">127:120</td><td rowspan="5">Device Capabilties</td><td colspan="2">This field describes the device capabilities</td></tr>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:6</td><td><b>Address Type:</b> This field describes the type of address contained in the device. Refer to the SMBus transport binding specification.</td></tr>
<tr><td>5:1</td><td>Reserved</td></tr>
<tr><td>0</td><td><b>PEC Supported:</b> All MCTP transactions shall include a Packet Error Code (PEC) byte. This field shall be set to one to indicate support for PEC.</td></tr>
<tr><td rowspan="5">119:112</td><td rowspan="5">Version / Revision</td><td colspan="2">This field is used to identify the UDID version and silicon revision.</td></tr>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:6</td><td>Reserved</td></tr>
<tr><td>5:3</td><td><b>UDID Version.</b> This field specifies the UDID version and shall be set to 001b</td></tr>
<tr><td>2:0</td><td><b>Silicon Revision ID:</b> This field is used to specify a vendor specific silicon revision level.</td></tr>
<tr><td>111:96</td><td>Vendor ID</td><td colspan="2">This field contains the PCI-SIG vendor ID for the Management Endpoint.</td></tr>
<tr><td>95:80</td><td>Device ID</td><td colspan="2">This field contains a vendor assigned device ID for the Management Endpoint.</td></tr>
<tr><td rowspan="8">79:64</td><td rowspan="8">Interface</td><td colspan="2">This field defines the SMBus version and the Interface Protocols supported.</td></tr>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>15:8</td><td>Reserved</td></tr>
<tr><td>7</td><td><b>ZONE.</b> This field shall be cleared to ‘0’.</td></tr>
<tr><td>6</td><td><b>IPMI.</b> This field shall be cleared to ‘0’.</td></tr>
<tr><td>5</td><td><b>ASF.</b> This field shall be set to ‘1’. Refer to the MCTP transport binding specification.</td></tr>
<tr><td>4</td><td><b>OEM.</b> This field shall be set to ‘1'.</td></tr>
<tr><td>3:0</td><td><b>SMBus Version.</b> This field shall be set to 4h or 5h which corresponds to SMBus Version 2.0 and 3.0 respectively.</td></tr>
<tr><td>63:48</td><td>Subsystem Vendor ID</td><td colspan="2">This field contains the PCI-SIG vendor ID for the Management Endpoint.</td></tr>
<tr><td>47:32</td><td>Subsystem Device ID</td><td colspan="2">This field contains a vendor assigned device ID for the Management Endpoint.</td></tr>
<tr><td rowspan="5">31:0</td><td rowspan="5">Vendor Specific ID</td><td colspan="2">This field contains a unique 30-bit static NVM storage device ID and is used to distinguish the NVM Subsystem UDID from the FRU Information Device UDID.</td></tr>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>31</td><td><b>UDID Type.</b> This field is used to distinguish the Management Endpoint UDID from the VPD UDID. A ‘1’ in this field indicates the Management Endpoint. A ‘0’ in this field indicates the FRU Information Device.</td></tr>
<tr><td>30</td><td>Reserved.</td></tr>
<tr><td>29:0</td><td><b>Unique NVM Storage Device ID:</b> This field contains a unique vendor assigned ID for the NVM Subsystem. The ID is different in each NVM Subsysteminstance and remains static during the life of the device.</td></tr>
</table>

Host platforms expecting to be used with one or more Management Endpoints (e.g., data center platforms and workstations) should isolate SMBus segments to avoid a Management Endpoint conflicting with the address of another SMBus device. An SMBus address conflict may occur when a Management Endpoint is used with platforms that do not isolate SMBus segments (e.g., some client platforms).

## 2.3 Error Handling

Physical layer errors are handled as specified by the corresponding physical layer specification and MCTP transport binding specification. There are no NVMe-MI physical layer specific error handling requirements beyond those outlined in these specifications.

# 3 Message Transport

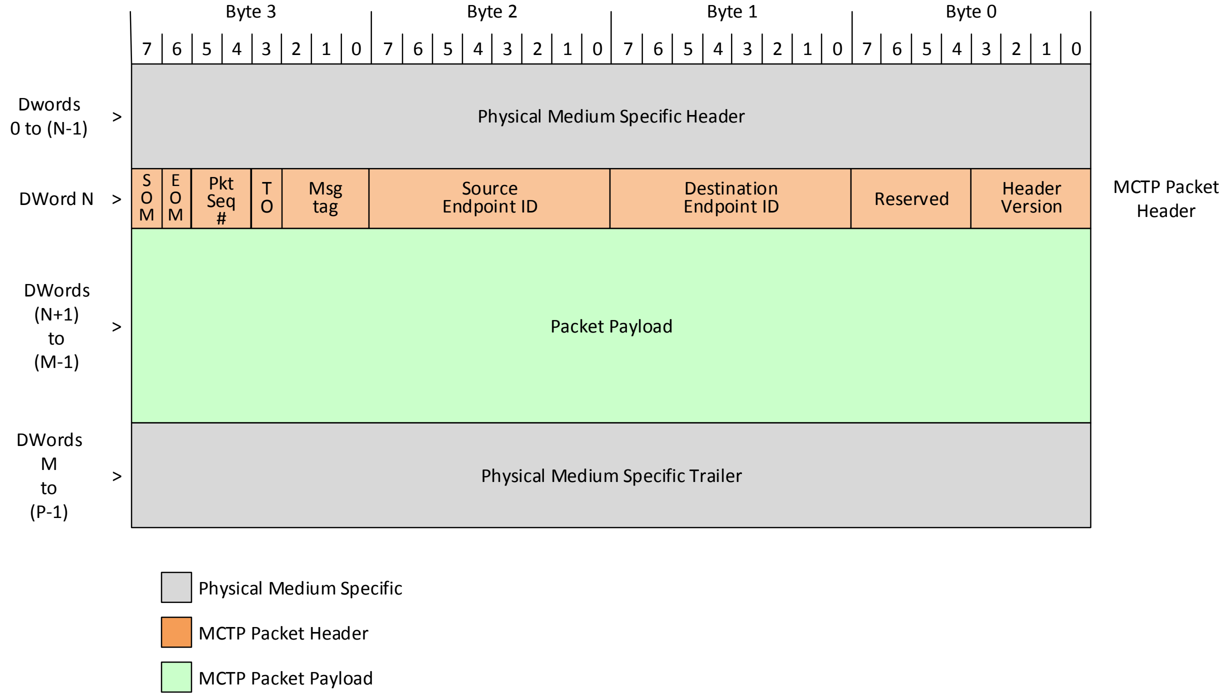
NVMe-MI utilizes MCTP as a reliable in-order message transport between a Management Controller and a Management Endpoint.

This section summarizes the NVMe-MI MCTP packet and message format. A Management Endpoint compliant to this specification shall implement all required behaviors detailed in the Management Component Transport Protocol (MCTP) Base Specification and corresponding transport binding specification in addition to the requirements outlined in this specification (e.g., the Message Integrity Check algorithm).

## 3.1 MCTP Packet

In MCTP, the smallest unit of data transfer is the MCTP packet. One or more packets are combined to create an MCTP message. A packet always contains at least 1 byte of payload but the total length shall never exceed the negotiated MCTP Transmission Unit Size. The format of an MCTP packet is shown in Figure 8.

**Figure 8: MCTP Packet Format**



MCTP specifications use big endian byte ordering while NVM Express specifications use little endian byte ordering. All figures in this specification are illustrated with little endian byte ordering. Note that this pictorial representation does not change the order that bytes are sent out on the physical layer.

The Physical Medium-Specific Header and Physical Medium-Specific Trailer are defined by the MCTP transport binding specification utilized by the port. Refer to the MCTP transport binding specifications.

The Management Component Transport Protocol (MCTP) Base Specification defines the MCTP packet header (refer to DSP0236 for field descriptions). The fields of an MCTP Packet are shown in Figure 9.

**Figure 9: MCTP Packet Fields**

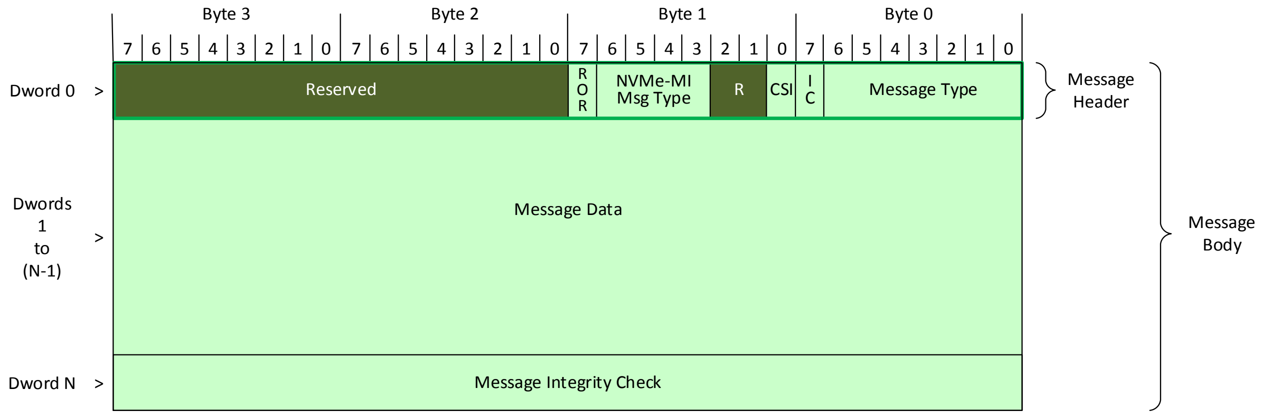
| Field Name | Field Size |
|---|---|
| Medium-Specific Header | varies |
| Header Version | 4 bits |
| Reserved | 4 bits |
| Destination Endpoint ID | 8 bits |
| Source Endpoint ID | 8 bits |
| Msg tag (Message Tag) | 3 bits |
| TO | 1 bit |
| Pkt Seq # | 2 bits |
| EOM | 1 bit |
| SOM | 1 bit |
| Packet Payload | varies |
| Medium-Specific Trailer | varies |

A compliant Management Endpoint shall implement all MCTP required features defined in the MCTP base specification. Optional features may be supported.

### 3.2 MCTP Messages

An MCTP message consist of the payload of one or more MCTP packets. The maximum sized message is 4224 bytes (4K + 128). Refer to the NVMe Management Messages over MCTP Binding Specification. Messages with lengths greater than 4224 are considered invalid messages. The format of an NVMe-MI MCTP message is shown in Figure 10.

**Figure 10: NVMe-MI MCTP Message**



### 3.2.1 Message Fields

The format of an NVMe-MI message consists of a Message Header in the first Dword, followed by the Message Data, and ends with the Message Integrity Check Dword as shown in Figure 10.

The Message Header contains a Message Type (MT) field and an Integrity Check (IC) field that are defined by the MCTP Base Specification. The Message Type field specifies the type of payload contained in the message body and is required to be set to 4h in all messages associated with NVMe-MI (refer to the MCTP IDs and Codes specification). The Integrity Check (IC) field indicates whether the message is covered by an overall MCTP Message Integrity Check. All NVMe-MI messages are protected by a 32-bit CRC computed over the message body contents. The IC field shall be set to ‘1’ in all NVMe-MI MCTP messages.

The Request or Response (ROR) bit in the Message Header specifies whether the NVMe-MI MCTP message is associated with a Request Message or a Response Message. The NVMe Message Type (NMIMT) field specifies whether the Request Message is a Control Primitive or a specific type of Command Message (refer to Figure 14). Finally the Command Slot Identifier (CSI) field specifies the Command Slot with which the message is associated. Refer to section 4 for additional information about Command Slots.

**Figure 11: NVMe-MI MCTP Message Fields**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7</td><td><b>Integrity Check (IC):</b> This field is defined by the MCTP Base Specification and indicates whether the MCTP message is covered by an overall MCTP Message Integrity Check.<br>All NVMe-MI messages are protected by a CRC and thus this bit shall be set to ‘1’ in all NVMe-MI messages.</td></tr>
<tr><td>6:0</td><td><b>Message Type (MT):</b> This field is defined by the MCTP Base Specification for the message type. This field shall be set to 4h in all NVMe-MI messages. Refer to MCTP IDs and Codes .</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7</td><td><b>Request or Response (ROR):</b> This field indicates whether the message is a Request Message or Response Message. This field is cleared to ‘0’ for Request Messages. This field is set to ‘1’ for Response Messages.</td></tr>
<tr><td>6:3</td><td><b>NVMe-MI Message Type (NMIMT):</b> This field specifies the NVMe-MI Message Type.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0h</td><td>Control Primitive – refer to section 4.4</td></tr>
<tr><td>1h</td><td>NVMe-MI Command – refer to section 5</td></tr>
<tr><td>2h</td><td>NVMe Admin Command – refer to section 6</td></tr>
<tr><td>3h</td><td>Reserved</td></tr>
<tr><td>4h</td><td>PCIe Command – refer to section 7</td></tr>
<tr><td>5h – Fh</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>2:1</td><td>Reserved</td></tr>
<tr><td>0</td><td><b>Command Slot Identifier (CSI)</b>: This field indicates the Command Slot with which the message is associated. For Request Messages this field indicates the Command Slot with which the Request Message is associated. For Response Messages, this field indicates the Command Slot associated with the Request Message with which the Response Message is associated.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0h</td><td>Command Slot 0</td></tr>
<tr><td>1h</td><td>Command Slot 1</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>3:2</td><td>Reserved</td></tr>
<tr><td>x-1:4</td><td><b>Message Data (DATA):</b> This field contains the NVMe-MI message payload. The format of this field depends on the NVMe-MI Message Type.</td></tr>
<tr><td>x+3:x</td><td><b>Message Integrity Check (MIC):</b> This field contains a CRC computed over the contents of the message. Refer to section 3.2.1.1.</td></tr>
</table>

### 3.2.1.1 Message Integrity Check

The Message Integrity Check field contains a 32-bit CRC computed over the contents of the NVMe-MI message. The 32-bit CRC used by NVMe-MI is CRC-32C (Castagnoli) which uses the generator polynomial 1EDC6F41h. The Message Integrity Check is calculated using the following Rocksoft™ Model CRC Algorithm parameters:

```
Name   : "CRC-32C"
```

```
Width  : 32
Poly   : 1EDC6F41h
Init   : FFFFFFFFh
RefIn  : True
RefOut : True
XorOut : FFFFFFFFh
Check  : E3069283h
```

When sending a message, the Message Integrity Check shall be calculated using the following procedure or a procedure that produces an equivalent result:

1. Initialize the CRC register to FFFFFFFFh. This is equivalent to inverting the lowest 32 bits of the NVMe-MI Message (Dword 0 in Figure 10).
2. Append 32 bits of 0's to the end of the Message Data to allow room for the Message Integrity Check (Dword N in Figure 10). This results in the Message Body shown in Figure 10 with the Message Integrity Check field cleared to 0h.
3. Map the bits in the Message Body from step 2 to the coefficients of the message polynomial M(x). Assume the length of M(x) is Y bytes. Bit 0 of byte 0 in the Message Body is the most significant bit of M(x), followed by bit 1 of byte 0, on through to bit 7 of byte Y - 1. Note that the bits within each byte are reflected (i.e., bit n of each byte is mapped to bit (7 - n) resulting in bit 7 to bit 0, bit 6 to bit 1, and so on).

**Figure 12: Message Integrity Check Example**

| | Message Body (Length = Y bytes) | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Byte 0 | | | | | | | | Byte 1 | | | | | | | | … | Byte Y - 1 | | | | | | | |
| M(x) = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | … | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

4. Divide the polynomial M(x) by the generator polynomial 1EDC6F41h to produce the 32-bit remainder polynomial R(x).
5. Reflect R(x) (i.e. bit n of each byte is mapped to bit (31 - n) resulting in bit 31 to bit 0, bit 30 to bit 1, and so on) to produce the polynomial R′(x).
6. Invert R′(x) to produce the polynomial R″(x).
7. Store R″(x) in the Message Integrity Check field of the Message Body.

Upon receipt of an NVMe-MI message, the Message Integrity Check may be validated as follows:

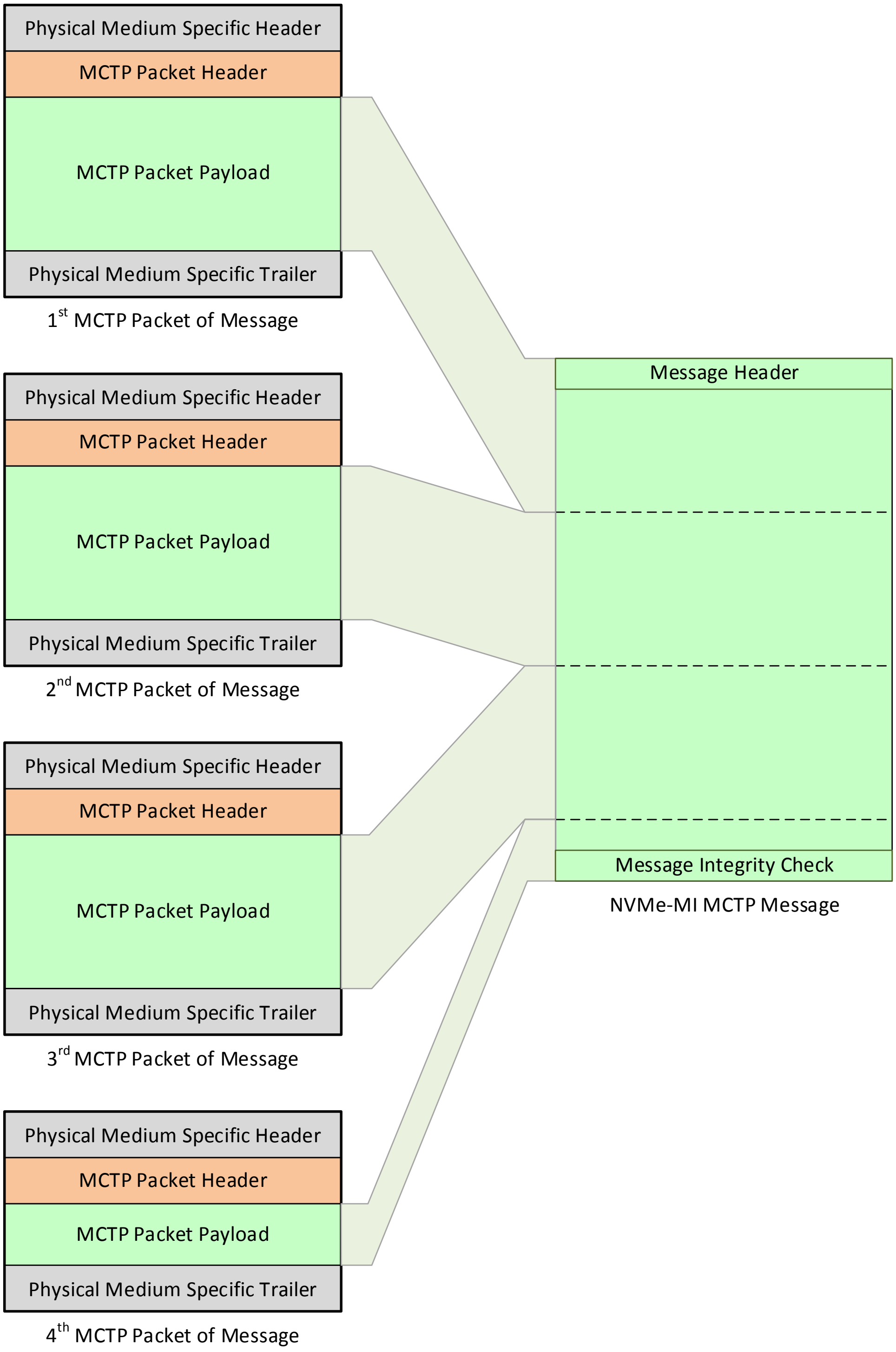
1. Save the received Message Integrity Check.
2. Initialize the CRC register to FFFFFFFFh. This is equivalent to inverting the lowest 32 bits of the NVMe-MI Message (Dword 0 in Figure 10).
3. Clear the Message Integrity Check field to 0h.
4. Map the bits in the Message Body to the coefficients of the message polynomial M(x) as described in step 3 in the Message Integrity Check calculation procedure above.
5. Divide the polynomial M(x) by the generator polynomial 1EDC6F41h to produce the 32-bit remainder polynomial R(x).
6. Reflect R(x) (i.e. bit n of each byte is mapped to bit (31 - n) resulting in bit 31 to bit 0, bit 30 to bit 1, and so on) to produce the polynomial R′(x).
7. Invert R′(x) to produce the polynomial R″(x).
8. Compare R″(x) from step 5 to the Message Integrity Check value saved in step 1. If both values are equal, the Message Integrity Check passes.

Refer to Appendix B for artificial messages and their corresponding Message Integrity Check values.

### 3.2.2 Packet Assembly into Messages

An NVMe-MI MCTP message may be split into multiple MCTP Packet Payloads and sent as a series of packets. An example message whose contents are split across four MCTP packets is shown in Figure 13. Refer to the MCTP Base Specification for packetization and message assembly rules.

**Figure 13: NVMe-MI MCTP Message Spanning Multiple MCTP Packets**


Physical Medium Specific Header
MCTP Packet Header
MCTP Packet Payload
Physical Medium Specific Trailer
1st MCTP Packet of Message
Physical Medium Specific Header
MCTP Packet Header
MCTP Packet Payload
Physical Medium Specific Trailer
2nd MCTP Packet of Message
Physical Medium Specific Header
MCTP Packet Header
MCTP Packet Payload
Physical Medium Specific Trailer
3rd MCTP Packet of Message
Physical Medium Specific Header
MCTP Packet Header
MCTP Packet Payload
Physical Medium Specific Trailer
4th MCTP Packet of Message
Message Header
Message Integrity Check
NVMe-MI MCTP Message

In addition to the requirements outlined in the MCTP Base Specification and transport binding specifications, the NVMe-MI Specification has the following additional requirements:

- With the exception of the last packet in a message, the MCTP Transmission Unit size of all packets in a given message shall be equal to the negotiated MCTP Transmission Unit Size.
- The MCTP Transmission Unit size of the last packet in a Request Message or Response Message (i.e., the one with the EOM bit set in the MCTP header) shall be the smallest size needed to transfer the MCTP Packet Payload for that Packet with no additional padding beyond any padding required by the physical medium-specific trailer.
- Once a complete NVMe-MI MCTP message has been assembled, the Message Integrity Check is verified. If the Message Integrity Check passes, then the message is processed. If the Message Integrity Check fails, then the message is discarded. Refer to 4.3.

## 3.3 Error Handling

The Management Endpoint shall drop (silently discard) packets for error conditions as specified in the MCTP Base Specification. Some example conditions which result in discarding packets include unexpected middle or end packets.

# 4 Message Processing Model

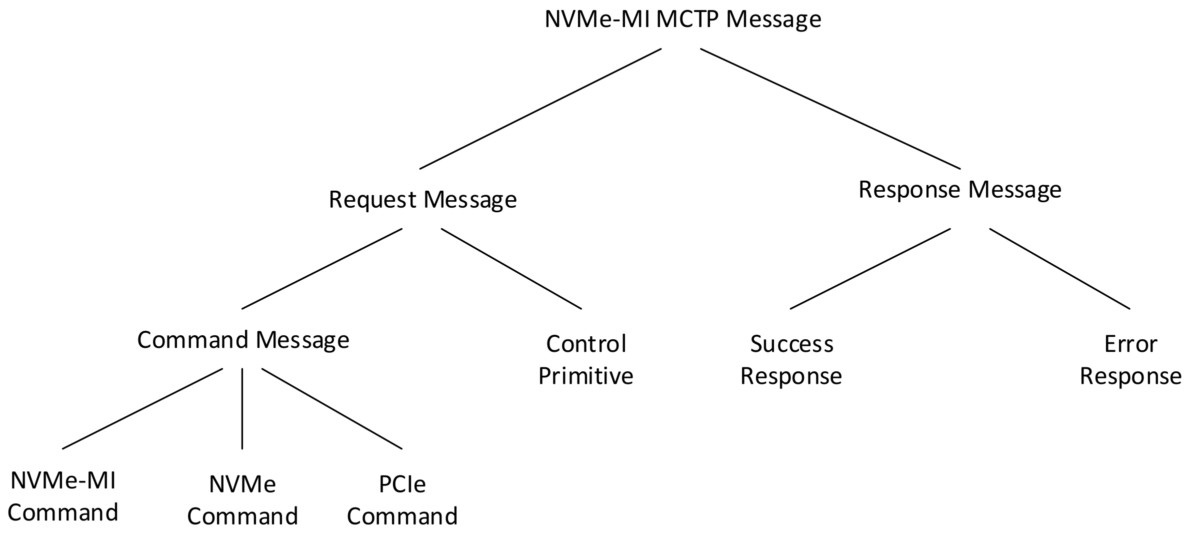
NVMe-MI utilizes a request and response processing model. A Management Controller sends a Request Message to a Management Endpoint, the Management Endpoint processes the Request Message, and when processing has completed, sends a Response Message back the Management Controller. Under no circumstances does a Management Endpoint generate an unsolicited Response Message (i.e., a Response Message that does not correspond to a previously received Request Message).

Figure 14 illustrates the taxonomy of NVMe-MI MCTP messages. A Request Message may be classified as a command or a Control Primitive. Commands specify an operation to be performed by the Management Endpoint and may be further classified as an NVMe-MI command, an NVMe Admin command, or a PCIe command. Control Primitives are used to affect the processing of a previously issued Command and are described in Section 4.4.

Unlike other NVMe-MI MCTP messages that may span multiple MCTP packets, messages containing a Control Primitive shall consist of exactly one MCTP packet.

A Response Message may be classified as a success response or an error response.

**Figure 14: NVMe-MI MCTP Message Taxonomy**



## 4.1 Request Messages

Request Messages are NVMe-MI messages that are generated by a Management Controller to send to a Management Endpoint.

Request Messages specify an action to be performed by the Management Endpoint. Request Messages are either Control Primitives (refer to 4.4) or Command Messages. The format of the message body for a Command Message is command set specific and is specified by the NMIMT field in the message header.

The NVMe Management Interface supports three command sets:

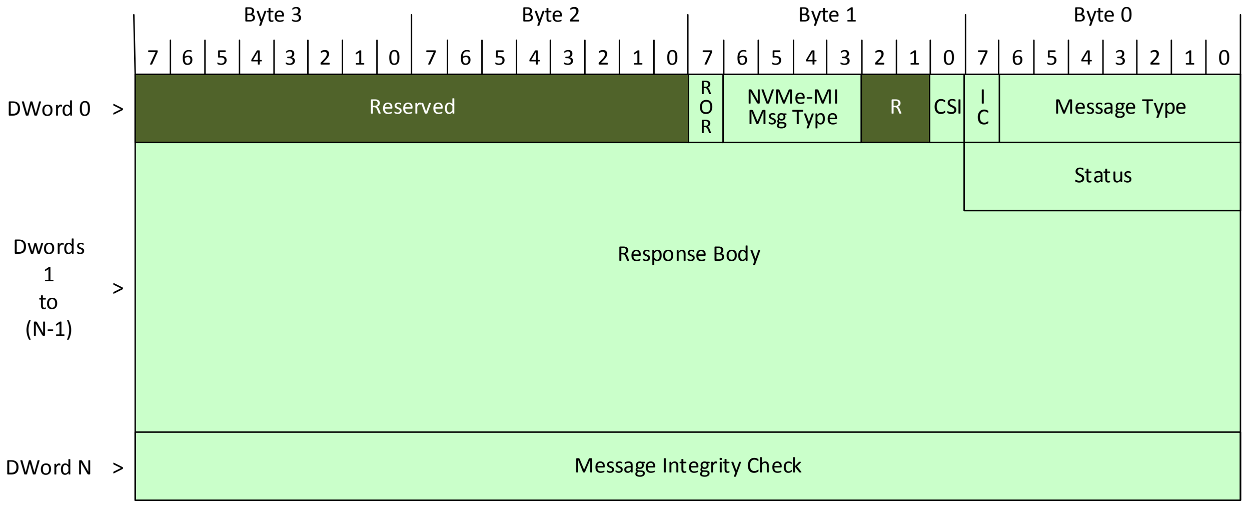
- The Management Interface command set is described in chapter 5.
- The NVM Express Admin command set is described in chapter 6.
- The PCIe command set is described in chapter 7.

## 4.2 Response Messages

Response Messages are NVMe-MI messages that are generated when a Management Endpoint completes processing of a previously issued Request Message.

The format of a Response Message is shown in Figure 15 and Figure 16. The first Dword contains the NVMe-MI message header. The Status field encodes the status associated with the Response Message. This is followed by the Response Body whose format is response status specific. Finally, the Response Message ends with the NVMe-MI Message Integrity Check field.

**Figure 15: Response Message Format**



The CSI field in the NVMe-MI Message Header specifies the Command Slot of the Request Message with which the Response Message is associated. The NVMe-MI Message Type (Msg Type) field contains the value from the same field in the corresponding Request Message.

**Figure 16: Response Message Fields**

| Byte | Description |
|---|---|
| 3:0 | **NVMe-MI Message Header:** Refer to Section 3.2. |
| 4 | **Status (STATUS):** This field indicates the status associated with the Response Message. Response Message Status values are summarized inFigure 17. |
| M-1:5 | **Response Body:** This field contains response specific fields whose format is dependent on the Status field. |
| M+3:M | **Message Integrity Check:** Refer to Section 3.2. |

Response Message Status values are summarized in Figure 17. A Response Message Status value of Success indicates that the corresponding Request Message completed successfully and that the Response Message is a success response. The format of the response body for a success response is dependent on the NVMe-MI message type and is described later in this specifiation.

A Response Message Status value other that Success indicates that an error occurred during processing of the corresponding command and that the response is an error response. The format of the response body is dependent on the Response Message Status value as shown in Figure 17. If multiple errors are present, a Management Endpoint may choose which error status to report.

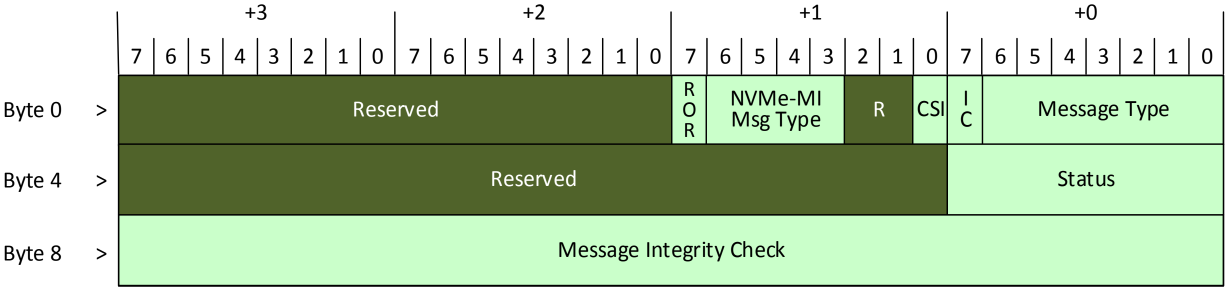
**Figure 17: Response Message Status Values**

| Value | Description | Error Reponse Format |
|---|---|---|
| 00h | **Success:** The command completed successfully. | Refer to 4.2.1 |
| 01h | **More Processing Required:** The command is in progress and requires more time to complete processing. When this Response Message Status value is used in a Response Message, a subsequent message contains the result of the Command Message. This Response Message Status Value shall not be sent more than once per Request Message. | Refer to 4.2.1 |
| 02h | **Internal Error:** The command could not be executed due to a vendor specific internal error. | Refer to 4.2.1 |
| 03h | **Invalid Command Opcode:** Invalid command opcode field value. | Refer to 4.2.1 |
| 04h | **Invalid Parameter:** Invalid command parameter field value. | Refer to 4.2.2 |
| 05h | **Invalid Command Size:** The Command Message body was larger or smaller than that expected by the command due to a reason other than too much or too little input data (e.g., the command did not contain all the required parameters or no input data was expected but the command message body is larger than that needed to contain the required parameters).<br>The expected command message body size is determined by the command opcode assuming no other errors are detected (e.g., invalid opcode or invalid field). | Refer to 4.2.1 |
| 06h | **Invalid Command Input Data Size:** The Command Message requires input data and contains too much or too little input data. | Refer to 4.2.1 |
| 07h | **Access Denied:** A command was prohibited from being executed due to a vendor specific protection mechanism. | Refer to 4.2.1 |
| 08h – 1Fh | Reserved | |
| 20h | **VPD Updates Exceeded:** More updates to the VPD are attempted than allowed. | Refer to 4.2.1 |
| 21h | **PCIe Inaccessible:** The PCIe functionality is not available at this time. | Refer to 4.2.1 |
| 22h – DFh | Reserved | |
| E0h – FFh | Vendor Specific | Vendor Specific |

### 4.2.1 Generic Error Response

A generic error response is generated for errors in which no additional information is provided beyond the Response Message Status. Bytes 5 to 7 are reserved. The format of a generic error response is shown in Figure 18.

**Figure 18: Generic Error Response**

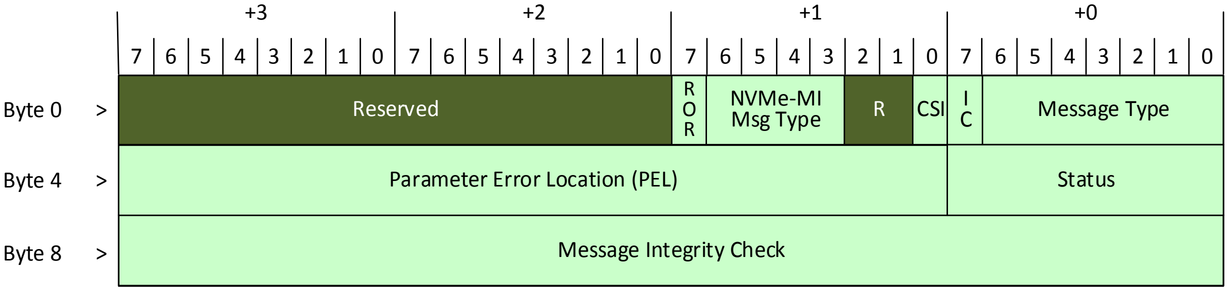


<table>
<tr><td></td><td colspan="8">+3</td><td colspan="8">+2</td><td colspan="8">+1</td><td colspan="8">+0</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0 ></td><td colspan="16">Reserved</td><td>ROR</td><td colspan="4">NVMe-MI Msg Type</td><td colspan="2">R</td><td>CSI</td><td>IC</td><td colspan="7">Message Type</td></tr>
<tr><td>Byte 4 ></td><td colspan="24">Reserved</td><td colspan="8">Status</td></tr>
<tr><td>Byte 8 ></td><td colspan="32">Message Integrity Check</td></tr>
</table>

### 4.2.2 Invalid Parameter Error Response

An invalid parameter error response is generated for error responses where the Status field is set to 03h (i.e., Invalid Parameter). The format of an invalid parameter error response is shown in Figure 19 and the response specific fields are summarized in Figure 20.

**Figure 19: Invalid Parameter Error Response**



<table>
<tr><td></td><td colspan="8">+3</td><td colspan="8">+2</td><td colspan="8">+1</td><td colspan="8">+0</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0 ></td><td colspan="16">Reserved</td><td>ROR</td><td colspan="4">NVMe-MI Msg Type</td><td colspan="2">R</td><td>CSI</td><td>IC</td><td colspan="7">Message Type</td></tr>
<tr><td>Byte 4 ></td><td colspan="24">Parameter Error Location (PEL)</td><td colspan="8">Status</td></tr>
<tr><td>Byte 8 ></td><td colspan="32">Message Integrity Check</td></tr>
</table>

**Figure 20: Invalid Parameter Error Response Fields**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>7:5</td><td><b>Parameter Error Location (PEL):</b> This field indicates the byte and bit of the request parameter within the Request Message that contains the first invalid parameter (i.e., the invalid parameter with the lowest byte and bit).<br>If the invalid parameter spans multiple bytes or bits, then the location indicates the first byte and bit of the parameter.<br>
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>23:8</td><td>Byte in the Request Message of the parameter that contained the error. Valid values are 0 to 4223.</td></tr>
<tr><td>7:3</td><td>Reserved</td></tr>
<tr><td>2:0</td><td>Bit in the Request Message of the parameter that contained the error. Valid values are 0 to 7.</td></tr>
</table>
</td></tr>
</table>

### 4.3 Command Processing Model

NVMe-MI utilizes Command Slots for command processing. Command Slots are logically used for MCTP NVMe-MI Request Message and Response Message assembly. Together with the request/response processing model, Command Slots provide a mechanism for message flow control. A Management Controller should not send a new Command Message to a Command Slot until the Response Message for the previously issued command to that Command Slot has been received.

Associated with each Management Endpoint are 2 Command Slots. Each Command Slot includes a state and a Pause flag (refer to 4.4.4).

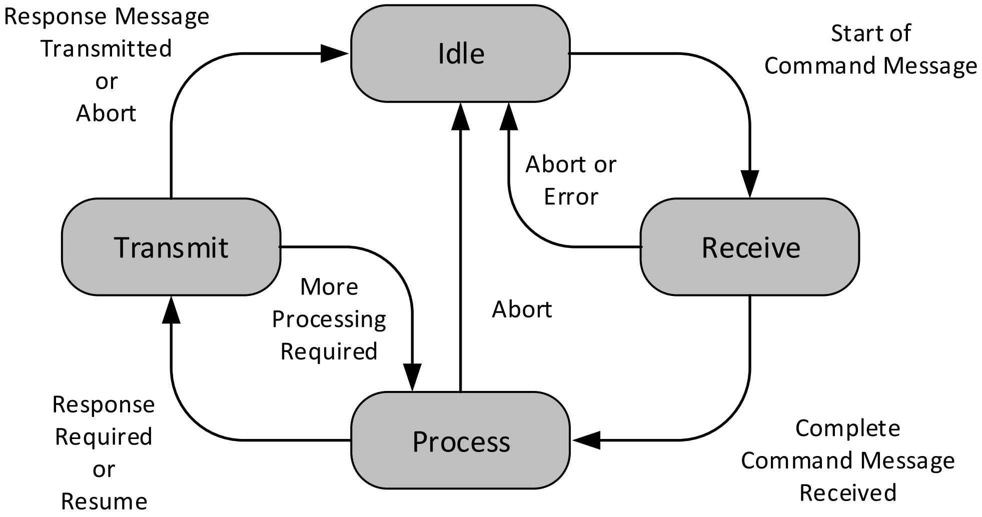
A Management Controller sends a Request Message to a Management Endpoint and targets a specific Command Slot in the Management Endpoint.

When a Management Endpoint receives MCTP packets for a Command Message that target a Command Slot, the packets are kept in a buffer associated with that Command Slot to be assembled. The Command Slot remains allocated to the Command Message until processing of the command has completed, the associated Response Message has been transmitted, and the Command Slot transitions back to the Idle state.

A Command Message is the only type of multi-packet MCTP NVMe-MI message that may be received by a Management Endpoint. The maximum number of Command Messages in flight to a Management Endpoint is equal to the number of Command Slots. The operation of each Command Slot is independent, allowing a Management Controller to have 2 independent streams of Command Messages to a Management Endpoint. The Command Message associated with each Command Slot are processed in parallel. If the NVM Subsystem implements multiple Management Endpoints, then command processing of each Management Endpoints occurs in parallel. A NVM Subsystem that implements *N* Management Endpoints may have up to 2*N* commands executing in parallel.

A Command Slot may be in 1 of 4 possible states. These states as well as valid state transitions are shown in Figure 21.

**Figure 21: Command Slot State Diagram**


Idle
Transmit
Receive
Process
Response Message Transmitted or Abort
Start of Command Message
Abort or Error
More Processing Required
Abort
Response Required or Resume
Complete Command Message Received

1. **Idle**: The state when there is no Command Message associated with the Command Slot. This is the default state of a Command Slot (e.g., following a reset). A Command Slot transitions from Idle to the Receive state when the first MCTP packet of a MCTP NVMe-MI command message is received (i.e., an MCTP packet with the SOM bit in the MCTP packet header set to ‘1’, the Message Type set to 4h, and the CSI field set to the corresponding Command Slot Identifier).

2. **Receive**: The state when the first packet of a Command Message has been received and the message is being assembled and/or validated. A Command Slot transitions from Receive to the Idle state when an Abort Control Primitive is received, an error is detected in message assembly (refer to 3.2.2), or the Message Integrity Check fails (refer to 3.2.1.1). A Command Slot transitions from Receive state to the Process state when a Command Message is assembled and the message integrity check is successful.

3. **Process**: The state when a Command Message is processed. Processing of a command consists of performing the actions specified by the command or aborting the command. A Command Slot transitions from Process to the Transmit state when a response is required (i.e., the Pause Flag is cleared to ‘0’ and either of the following are true: all processing of the command has completed or command processing is expected to exceed the corresponding transport binding specification response timeout). A Command Slot transitions from the Process state to the Idle state due to an Abort Control Primitive (refer to 4.4.3).

4. **Transmit**: The state in which a Response Message for the Command Message is transmitted to the Management Controller. A Command Slot transitions from the Transmit to the Idle state once the entire MCTP message associated with the response to the command has been transmitted on the physical medium or due to an Abort Control Primitive (refer to 4.4.3). If command processing did not complete in the Process state, then the Management Endpoint transmits a response with status More Processing Required and the Command Slot transitions back to the Process state.

Receiving a new Command Message "start" packet (packet with SOM = 1b) to the same Command Slot while a Command Message is being assembled (i.e., in the Receive state) terminates the original message assembly. All data for the terminated Command Message is discarded. The newly received start packet is not dropped, but instead it begins a new message assembly. This is considered receiving a Command Message to a non-Idle Command Slot (CMNICS). Refer to section 4.4.4.

If a Command Message packet is received when the corresponding Command Slot is in the Process or Transmit state, then the Management Endpoint discards the Command Message packet without a response. This is also considered receiving a Command Message to a non-Idle Command Slot (CMNICS). Refer to section 4.4.4.

## 4.4 Control Primitives

Control Primitives are Request Messages sent from a Management Controller to a Management Endpoint to affect the command processing flow. Control Primitives may target a Command Slot. Unlike Command Messages, Control Primitives may be sent while the Command Slot is any state and are processed immediately by the Management Endpoint. Unless otherwise indicated, Control Primitives do not change the state of the Command Slot.

The format of a Control Primitive is shown in Figure 22 and the fields are described in Figure 23.

**Figure 22: Control Primitive Request Message Format**

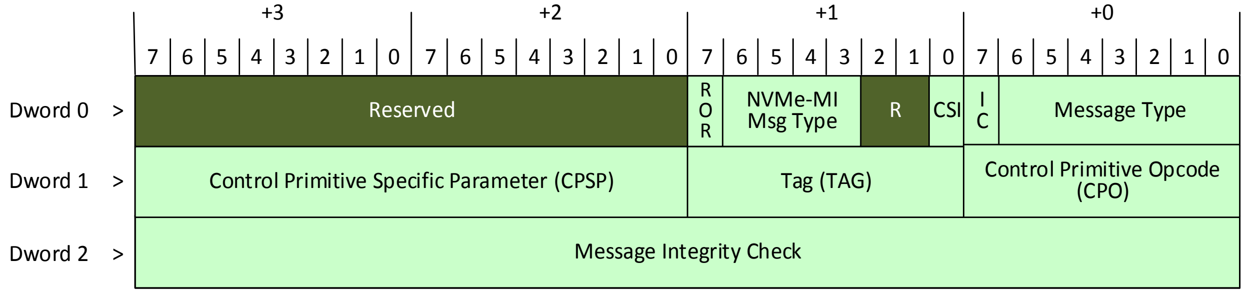


**Figure 23: Control Primitive Request Message Fields**

| Byte | Description |
|---|---|
| 03:00 | **NVMe-MI Message Header**: Refer to Section 3.2. |
| 04 | **Control Primitive Opcode (CPO):** This field specifies the opcode of the Control Primitive to be executed. Refer to Figure 24. |
| 05 | **Tag (TAG):** This field contains an opaque value that is sent from the Management Controller in the Control Primitive Request Message and returned by the Management Endpoint in to the associated response. A Management Controller may use any value in this field. |
| 07:06 | **Control Primitive Specific Parameter (CPSP):** This field is used to to pass Control Primitive specific parameter information. |

**Figure 24: Opcodes for Control Primitives**

| Opcode | O/M[1] | Command |
|---|---|---|
| 00h | M | Pause |
| 01h | M | Resume |
| 02h | M | Abort |
| 03h | M | Get State |
| 04h | M | Replay |
| 05h - EFh | | Reserved |

| Opcode | O/M[1] | Command |
|---|---|---|
| F0h - FFh | O | Vendor Specific |
| NOTES:<br>1. Optional or Mandatory; O – optional and M - mandatory | | |

The format of a success response associated with a Control Primitive is shown in Figure 25 and the fields are described in Figure 26.

**Figure 25: Control Primitive Success Response Message Format**

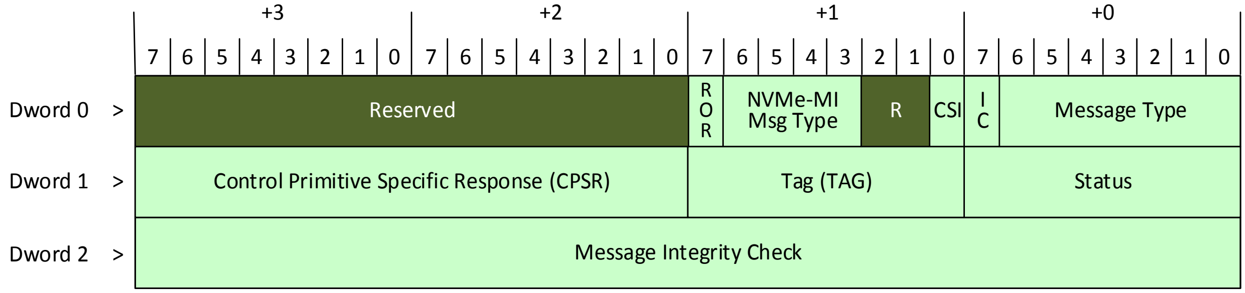


**Figure 26: Control Primitive Success Response Message Fields**

| Byte | Description |
|---|---|
| 03:00 | **NVMe-MI Message Header**: Refer to Section 3.2. |
| 04 | **Status:** Refer to Section 4.2. |
| 05 | **Tag (TAG):** This field contains an opaque value that is passed by the Management Endpoint from the Control Primitive to the associated Response Message. The Response Message contains the same value in this field as the corresponsing Request Message. |
| 07:06 | **Control Primitive Specific Response (CPSR):** This field is used to return Control Primitive specific status. |

A Management Endpoint transmits a Response Message to the Management Controller when the actions associated with that Control Primitive have completed.

Unlike Command Messages, a Management Controller may issue a Control Primitive to a Command Slot without waiting for a response for previously issued Control Primitives to that Command Slot. If multiple Control Primitives are sent without waiting for responses from the Management Endpoint, only the actions and response associated with the last Control Primitive are guaranteed (i.e., the actions associated with previously issued but unacknowledged Control Primitives may or may not be performed and the Response Messages for previously issued but unacknowledged Control Primitives may or may not be transmitted). Receipt of a Control Primitive never corrupts a previous Control Primitive associated with the Command Slot. The Response Message is either entirely transmitted or discarded.

The TAG field is an opaque value copied from the Control Primitive Request Message into the Response Message. By using unique TAG values it is possible for the Management Controller to link Response Messages with Request Messages.

### 4.4.1 Pause

The Pause Control Primitive is used to suspend response transmission and suspend the timeout waiting for packet for both Command Slots in a Management Endpoint. The CSI field in a Pause Control Primitive is not used and shall be cleared to ‘0’.

Associated with each Command Slot is a Pause Flag that determines whether the slot is ‘paused.’ The Pause Flag status is included with a success Response Message, and may also be read using the Get State primitive.

The CPSP field for the Pause primitive is reserved.

The format of the CPSR field in the Control Primitive success Response Message is shown in Figure 27.

**Figure 27: Pause Control Primitive Success Response Message Fields**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>08:07</td><td><b>Control Primitive Specific Response (CPSR):</b> This field is used to return Control Primitive specific status.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>15:02</td><td>Reserved</td></tr>
<tr><td>01</td><td><b>Pause Flag Status Slot 1 (PFSS1):</b> This field indicates whether or not Command Slot 1 is paused after completing the Pause primitive. A ‘1’ in this field indicates the Command Slot is paused. A ‘0’ in this field indicates the Command Slot is not paused.</td></tr>
<tr><td>00</td><td><b>Pause Flag Status Slot 0 (PFSS0):</b> This field indicates whether or not Command Slot 0 is paused after completing the Pause primitive. A ‘1’ in this field indicates the Command Slot is paused. A ‘0’ in this field indicates the Command Slot is not paused.</td></tr>
</table>
</td></tr>
</table>

The result of a Pause Control Primitive on a Command Slot is dependent on the state of the Command Slot when the Pause Control Primitive is received, as described below:

> **Idle**: The Pause primitive has no effect, and the Pause Flag is not changed (i.e., remains cleared to ‘0’). Refer to 4.4.4.
>
> **Receive**: The Pause primitive sets the Pause Flag to ‘1’ (refer to 4.4.4) and alerts the Management Endpoint that remaining MCTP packets associated with the command may be delayed. Further packets sent to this Command Slot while the Pause Flag is set are received normally.
>
> **Process**: The Pause primitive sets the Pause Flag to ‘1’ (refer to 4.4.4) causing the Command Slot to remain in the Process state until a Resume Control Primitive is received. Pause has no effect on the command processing in the Command Slot. Though command processing may complete, the Command Slot shall not transition to the Transmit state.
>
> **Transmit**: The Pause primitive sets the Pause Flag to ‘1’ (refer to 4.4.4) suspending transmission of MCTP response packets associated on a packet boundary with the Command in the Command Slot.

The Management Endpoint shall transmit a Response Message with success status after receiving the Pause primitive. It is not an error to issue a Pause Control Primitive when a Command Slot is already paused.

While the Pause Flag is set, the Management Endpoint disables the timeout waiting for packet timer and does not transmit responses to commands. The timeout waiting for a packet is the lesser of 100ms or the time defined in the appropriate MCTP transport binding specification. The Management Controller should not send commands while a Management Endpoint is paused.

### 4.4.2 Resume

The Resume Control Primitive is used to resume from a paused state. This is the complement to the Pause

Control Primitive.

Like the Pause Control Primitive, the Resume Control Primitive affects both slots and the CSI field in a Resume Control Primitive shall be cleared to ‘0’. If a Command Slot was not paused before receiving the Resume primitive, the Resume primitive completes successfully and has no effect.

The CPSP field for the Resume primitive is reserved. The CPSR field in the Control Primitive success Response Message is reserved.

The result of a Resume Control Primitive is based on the state of a Command Slot when the Resume Control Primitive is received, as described below:

**Idle**: The Resume primitive has no effect.

**Receive**: The Resume primitive alerts the Management Endpoint that transmission of any remaining MCTP packets associated with the command is resuming. The Pause Flag is cleared to ‘0’ (refer to 4.4.4).

**Process**: The Resume primitive allows a previously paused Command Slot to transition to the Transmit state and starts transmitting a response after responding to the Resume primitive. The Pause Flag is cleared to ‘0’ (refer to 4.4.4).

**Transmit**: The Management Endpoint resumes transmission of the response corresponding to the command associated with that slot after responding to the Resume primitive. The Pause Flag is cleared to ‘0’ (refer to 4.4.4).

The Management Endpoint shall transmit a Control Primitive Response Message with success status after receiving the Resume primitive.

### 4.4.3 Abort

The Abort Control Primitive is used to re-initialize a Command Slot to the Idle state, clear the Pause Flag associated with that Command Slot, and attempt to abort command processing associated with that Command Slot.

Aborting a Command Message shall have no effect on the other Command Slot of the Management Endpoint, other Management Endpoints, or NVMe Controllers in the NVM Subsystem. Subsequent command processing in the Command Slot is not affected by the Abort.

A Management Controller may issue an Abort primitive to clean-up resources associated with a Command Slot in an unknown state.

The CPSP field for the Abort primitive is reserved. The format of the CPSR field in the Control Primitive success Response Message is shown in Figure 28.

**Figure 28: Abort Control Primitive Success Response Message Fields**

| Byte | Description |
|---|---|

<table>
<tr><td>08:07</td><td><b>Control Primitive Specific Response (CPSR):</b> This field is used to return Control Primitive specific status.<br>
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>15:02</td><td>Reserved</td></tr>
<tr><td>01:00</td><td><b>Command Processing Abort Status (CPAS):</b> This field indicates the effect of the Abort primitive on the processing of the Command Message associated with the Command Slot.<br><br>0h – Command aborted after processing completed or no command to abort.<br>1h – Command aborted before processing began<br>2h – Command processing partially completed.<br>3h – Reserved</td></tr>
</table>
</td></tr>
</table>

The result of an Abort primitive is based on the state of the specified Command Slot when the Abort primitive is received, as described below:

**Idle**: The Abort primitive has no effect. The Management Endpoint shall transmit a Response Message with success status and the CPAS field cleared to 0h.

**Receive**: The Management Endpoint discards the contents of the Command Slot and transitions to the Idle state. The Management Endpoint shall transmit a Response Message with success status and the CPAS field set to 1h.

**Process**: The Abort primitive causes processing of the command in the Command Slot to be aborted.

- If the Abort primitive was received before command processing started, the Management Endpoint discards the contents of the Command Slot and transitions to the Idle state. The Management Endpoint shall transmit a success Response Message and the CPAS field set to 1h.

- If the Abort primitive was received while the command is being processed, the Management Endpoint discards the contents of the Command Slot and transitions to the Idle state. The Management Endpoint attempts to abort the command.

    - If the command is aborted and had no effect on the NVM Subsystem, then the Management Endpoint shall transmit a success Response Message and the CPAS field set to 1h.
    - If the Management Endpoint is not able to abort the command, then the Management Endpoint shall transmit a success Response Message and set the CPAS field to 2h.
    - If the command has completed processing (e.g., the Management Endpoint is paused), then the Management Endpoint shall transmit a success Response Message and the CPAS field is cleared to 0h.

**Transmit**: The Management Endpoint discards the contents of the Command Slot and transitions to the Idle state. The Management Endpoint transmits a Response Message with success status and the CPAS field cleared to 0h.

It is not an error to issue an Abort Control Primitive to a slot that is paused. The state of slot is reinitialized clearing the Pause Flag.

### 4.4.4 Get State

The Get State Control Primitive is used to check the state of a Command Slot.

The format of the CPSP field in the Control Primitive Request Message is shown in Figure 29.

**Figure 29: Get State Control Primitive Request Message Fields**

| Byte | Description |
|---|---|

<table>
<tr>
<td>08:07</td>
<td><b>Control Primitive Specific Parameter (CPSP):</b> This field is used to to pass Control Primitive specific parameter information.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>15:01</td><td>Reserved</td></tr>
<tr><td>00</td><td><b>Clear Error State Flags (CESF):</b> This field specifies whether or not to clear the error state flags when completing this command.</td></tr>
</table>
</td>
</tr>
</table>

The Management Endpoint shall transmit a Response Message with success status after receiving the Get State primitive. The format of the CPSR field in the Control Primitive success Response Message is shown in Figure 30.

Bits 04 through 13 are global for the Management Endpoint and indicate MCTP transport errors that have occurred. Refer to the MCTP Base Specification section for Dropped Packets and Dropped Messages for details on the errors.

**Figure 30: Get State Control Primitive Success Response Message Fields**

| Byte | Description |
|---|---|

| | | |
|---|---|---|
| 08:07 | **Control Primitive Specific Response (CPSR):** This field is used to return Control Primitive specific status. | |
| | **Bits** | **Description** |
| | 15 | **Pause Flag (PFLG):** This field indicates whether or not the Command Slot is paused. A ‘1’ in this field indicates the Command Slot is paused. A ‘0’ in this field indicates the Command Slot is not paused.<br>While the Pause Flag is set, the Management Endpoint disables the timeout waiting for packet timer, as defined in the MCTP Base Specification, for the Command Slot and does not transmit responses to Command Messages. |
| | 14 | **NVM Subsystem Reset Occurred (NSSRO):** This field indicates when an NVM Subsystem Reset occurs while main power is applied. This field is set to ‘1’ if the last occurrence of an NVM Subsystem Reset occurred while main power was applied to the NVM Subsystem. This field is cleared to ‘0’ following a power cycle and following a Get State primitive with the CESF field set to ‘1’. |
| | 13 | **Bad Packet or Other Physical Layer (BPOPL):** This field is set to ‘1’ if a packet sent to the Management Endpoint failed a transport specific packet integrity check since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 12 | **Bad, Unexpected, or Expired Message Tag (BUEMT):** This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 11 | **Out-of-Sequence Packet Sequence Number (OSPSN):** This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 10 | **Unexpected Middle or End of Packet (UMEP):** This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 09 | **Incorrect Transmission Unit (ITU):** This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 08 | **Unknown Destination ID (UDSTID):** This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 07 | **Bad Header Version (BHVS):** This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 06 | **Unsupported Transmission Unit (UTUNT):** This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 05 | **Timeout Waiting for a Packet (WPTT)**: This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 04 | **Bad Message Integrity Check Error (TMICE):** This field is set to ‘1’ if the Management Endpoint detected an error of this type (refer to the MCTP Base Specification) since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 03 | **Command Message to non-Idle Command Slot (CMNICS):** This field is set to ‘1’ if the Management Endpoint received a Command Message packet while the Command Slot is not in the Idle state since the last time Get State primitive was executed with the CESF field set to ‘1’. |
| | 02 | Reserved |
| | 01:00 | **Slot State (SSTA):** This field indicates the current state of the Command Slot. |

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td></td><td><table><tr><td></td><td><table><tr><th>Value</th><th>Description</th></tr><tr><td>0h</td><td>Idle</td></tr><tr><td>1h</td><td>Receive</td></tr><tr><td>2h</td><td>Process</td></tr><tr><td>3h</td><td>Transmit</td></tr></table></td></tr></table></td></tr>
</table>

### 4.4.5 Replay

The Replay Control Primitive is used to retransmit the Response Message for the last Command Message processed in a Command Slot.

The format of the CPSP field in the Control Primitive Request Message is shown in Figure 15.

**Figure 31: Replay Control Primitive Request Message Fields**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>08:07</td><td><b>Control Primitive Specific Parameter (CPSP):</b> This field is used to to pass Control Primitive specific parameter information.<table><tr><th>Bits</th><th>Description</th></tr><tr><td>15:08</td><td>Reserved</td></tr><tr><td>07:00</td><td><b>Response Replay Offset (RRO):</b> This field specifies the starting packet number from which the Response Message associated with the last Command Message processed in the Command Slot should be replayed.<br><br>This is a 0’s based value. When this field is cleared to ‘0’, the first packet of the associated Response Message is the first packet replayed.<br><br>If this field specifies an offset that is beyond the length of the Response Message, then processing of the Control Primitive is aborted and the Management Endpoint transmits an Invalid Parameter Error Response Message.</td></tr></table></td></tr>
</table>

The format of the CPSR field in the Control Primitive success Response Message is shown in Figure 32.

**Figure 32: Replay Control Primitive Success Response Message Fields**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>08:07</td><td><b>Control Primitive Specific Response (CPSR):</b> This field is used to return Control Primitive specific status.<table><tr><th>Bit</th><th>Description</th></tr><tr><td>15:01</td><td>Reserved</td></tr><tr><td>00</td><td><b>Response Replay (RR):</b> This bit indicates if a previous Response Message is retransmitted. This field is set to ‘1’ if the requested Response Messageis retransmitted by the Management Endpoint. This field is cleared to ‘0’ if the requested Response Message is not retransmitted.</td></tr></table></td></tr>
</table>

The result of a Replay primitive is based on the state of the specified Command Slot when the Replay primitive is received, as described below:

**Idle**: The Replay primitive requests retransmission of the completion at the offset specified by the RRO field if such a completion is available.

- If the Replay primitive was received following an Abort primitive or a reset (refer to 9.3) before any Command Messages are processed, then there is no Response Message available to retransmit. The Management Endpoint shall transmit a Response Message with success status with the RR field cleared to '0'.

- If the Replay primitive was received following the processing of one or more Command Messages, then the Management Endpoint shall transmit a Response Message with success status with the RR field set to '1'. The Management Endpoint transmits the MCTP packets associated with the requested Response Message after the Control Primitive success response.

**Receive**: The Management Endpoint transmits a Response Message with success status with the RR field cleared to '0'.

**Process**: The Replay primitive requests retransmission of the last response transmitted for the command in this Command Slot.

- If a Response Message has not been transmitted for the Command Message (i.e., the slot never entered the Transmit state for the Command Message), then the Management Endpoint transmits a Response Message with success status and the RR field cleared to '0'.

- If a Response Message has been transmitted for the Command Message (i.e., a Response Message was transmitted indicating that more processing was required), then the Management Endpoint transmits a Response Message with success status with the RR field set to '1'. The Management Endpoint retransmits the response indicating that more processing is required.

**Transmit**: The Management Endpoint stops transmitting response packets for the Command Slot. The Management Endpoint transmits a Response Message with success status with the RR field set to '1'. The Management Endpoint transmits the MCTP packets associated with the current command response after the Control Primitive success response. The Command Slot remains in the Transmit state until retransmission is complete.

It is not an error to issue a Replay primitive to a Command Slot that is paused. The response is retransmitted even if the Command Slot was paused (i.e., there is an implicit Resume primitive affecting both Command Slots when processing the Replay primitive) at any time during the response including before the first packet was transmitted. After successful completion of the Replay primitive, neither Command Slot is paused.

## 4.5 Error Handling

This section describes error handling specific to the NVMe-MI message processing model.

### 4.5.1 Command Timeouts

MCTP defines a maximum response time for MCTP control messages (refer to the appropriate MCTP transport binding specification).

If a Management Endpoint determines that command processing may not complete within the lesser of 100ms or the request-to-response time specified in the appropriate MCTP transport binding specification, the Management Endpoint shall utilize the More Processing Required response mechanism. The Response Message from the Management Endpoint may only be delayed beyond this timeout while the transport is busy or unavailable.

A Management Endpoint should only use the More Processing Required response for commands that are expected to take longer than the required time (e.g., Format NVM). Implementations are strongly discouraged from using this response while processing normal commands.

#### 4.5.2 Control Primitive Timeouts

A Management Endpoint shall attempt to respond to a Control Primitive within the lesser of 100ms or the request-to-response time specified in the appropriate MCTP transport binding specification. The Response Message from the Management Endpoint may only be delayed beyond this timeout while the transport is busy or unavailable.

# 5 Management Interface Command Set

The Management Interface Command Set defines the Command Messages that may be submitted by a Management Controller when the NMIMT value is set to NVMe-MI Command.

The MCTP Message structure with all fields that are common to all MCTP Messages are defined for commands in section 3.2. The Response Message structure for Management Interface Command Set is defined in section 4.2. The Message Body for Management Interface Commands is shown in Figure 34. Command specific fields for the Management Interface command set are defined in this section.

**Figure 33: Management Interface Command Request Message Format**

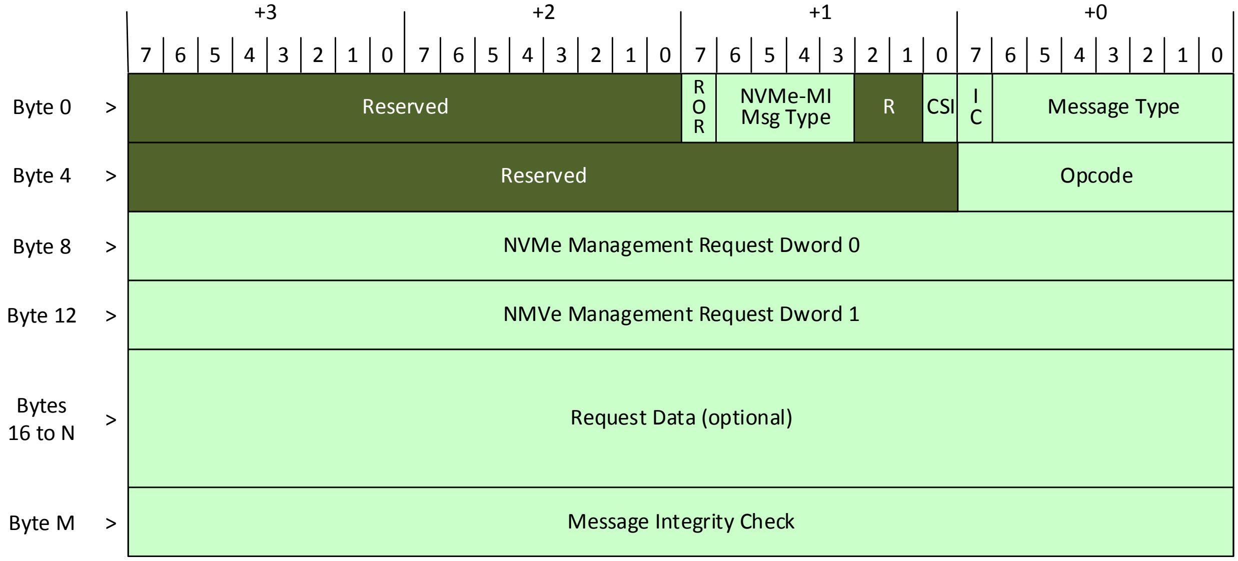


**Figure 34: Management Interface Request Message Description**

| Byte | Description |
|---|---|
| 03:00 | **NVMe-MI Message Header**: Refer to 3.2. |
| 04 | **Opcode (OPC):** This field specifies the opcode of the NVMe Management Interface command to be executed. Refer to Figure 35. |
| 07:05 | Reserved |
| 11:08 | **NVMe Management Dword 0 (NMD0):** This field is command specific Dword 0. |
| 15:12 | **NVMe Management Dword 1 (NMD1):** This field is command specific Dword 1. |
| N:16 | **Request Data** (Optional) |
| M+3:M | **Message Integrity Check (MIC):** Refer to 3.2. |

The Request Data field is an optional field included in some Management Interface commands. If the size of the Request Data does not match the specified Data Length of the Command Message, then the Management Endpoint responds with a generic error response and Invalid Command Input Data Size status.

Figure 35 defines the Management Interface Command Set opcodes.

**Figure 35: Opcodes for Management Interface Commands**

| Opcode | O/M[1] | Command |
|---|---|---|
| 00h | M | Read NVMe-MI Data Structure |
| 01h | M | NVM Subsystem Health Status Poll |
| 02h | M | Controller Health Status Poll |
| 03h | M | Configuration Set |
| 04h | M | Configuration Get |
| 05h | M | VPD Read |
| 06h | M | VPD Write |
| 07h | M | Reset |
| 08h – BFh | - | Reserved |
| C0h – FFh | O | Vendor specific |
| NOTES:<br>1. O/M definition: O = Optional, M = Mandatory. | | |

**Figure 36: Management Interface Command Response Message Format**

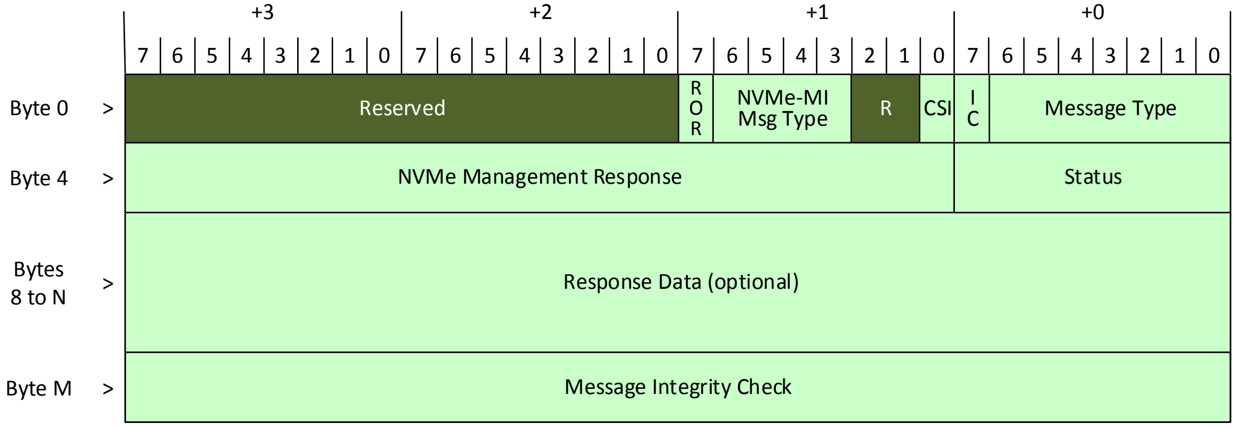


**Figure 37: Management Interface Response Message Description**

| Byte | Description |
|---|---|
| 03:00 | **NVMe-MI Message Header**: Refer to 3.2. |
| 04 | **Status:** This field indicates the status of the NVMe-MI command. Refer to 4.2. |
| 07:05 | **NVMe Management Response:** This field is command specific. |
| N:16 | **NVMe Response Data** (optional) |
| M+3:M | **Message Integrity Check:** Refer to 3.2. |

## 5.1 Configuration Get

The Configuration Get command allows the Management Controller to read the current configuration of a Management Endpoint.

The command uses NVMe Management Dwords 0 and 1. The format of NVMe Management Dwords 0 and 1 are shown in Figure 38 and Figure 39 respectively. There is no Request Data included in a Configuration

Get command.

**Figure 38: Configuration Get – NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:08 | Configuration specific |
| 07:00 | **Configuration Identifier:** This field specifies the identifier of the Configuration that is being read. Refer to Figure 40. |

**Figure 39: Configuration Get – NVMe Management Dword 1**

| Bit | Description |
|---|---|
| 31:00 | Configuration specific |

NVMe-MI Configurations are listed in Figure 40. Specifying a reserved identifier in the Configuration Identifier field causes the command to complete with an Invalid Parameter error status.

**Figure 40: NVMe Management Interface Configuration Identifiers**

| Configuration Identifier | O/M[1] | Description |
|---|---|---|
| 00h | | Reserved |
| 01h | M | SMBus/I2C Frequency |
| 02h | M | Health Status Change |
| 03h | M | MCTP Transmission Unit Size |
| 04h - BFh | | Reserved |
| C0h - FFh | O | Vendor Specific |
| NOTES:<br>1. O/M definition: O = Optional, M = Mandatory. | | |

The NVMe Management Response field is configuration specific.

### 5.1.1 SMBus/I2C Frequency (Configuration Identifier 01h)

The SMBus/I2C Frequency configuration indicates the current frequency of the SMBus port, if applicable.

The configuration specific fields in NVMe Management Dword 0 are shown in Figure 41. The configuration specific fields in NVMe Management Dword 1 are reserved. The current SMBus/I2C Frequency configuration is returned in the NVMe Management Response field as shown in Figure 42.

**Figure 41: SMBus/I2C Frequency – NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:24 | **Port Identifier:** This field specifies the port whose SMBus/I2C Frequency is indicated. |
| 23:08 | Reserved |
| 07:00 | **Configuration Identifier:** This field specifies the identifier of the Configuration that is being read. Refer to Figure 40. |

**Figure 42: SMBus/I2C Frequency – NVMe Management Response**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>23:04</td><td>Reserved</td></tr>
<tr><td>03:00</td><td><b>SMBus/I2C Frequency:</b> The current frequency of the SMBus/I2C. The default value for this field following a reset or power cycle is 1h, if SMBus is supported.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0h</td><td>SMBus is not supported or is disabled</td></tr>
<tr><td>1h</td><td>100 kHz</td></tr>
<tr><td>2h</td><td>400 kHz</td></tr>
<tr><td>3h</td><td>1 MHz</td></tr>
<tr><td>4h - Fh</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

### 5.1.2 Health Status Change (Configuration Identifier 02h)

The Health Status Change configuration is used to clear the selected status bits in the Composite Controller Status field using Configuration Set. A Management Controller should not use Configuration Get for this Configuration Identifier.

The configuration specific fields in NVMe Management Dwords 0 and 1 are reserved. A Management Endpoint shall complete a Configuration Get command on this Configuration Identifier with a Success Response Message. The NVMe Management Response field is reserved and there is no Response Data.

### 5.1.3 MCTP Transmission Unit Size (Configuration Identifier 03h)

The MCTP Transmission Unit Size configuration indicates the current MCTP Transmission Unit Size of the port Identifier specified in Dword 0.

The configuration specific fields in NVMe Management Dword 0 are shown in Figure 43. The configuration specific fields in NVMe Management Dword 1 are reserved. The current Transmission unit size of the specified port is returned in the NVMe Management Response field as shown in Figure 44.

**Figure 43: MCTP Transmission Unit Size – NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:24 | **Port Identifier:** This field specifies the port whose MCTP Transmission Unit Size is indicated. |
| 23:08 | Reserved |
| 07:00 | **Configuration Identifier:** This field specifies the identifier of the Configuration that is being read. Refer to Figure 40. |

**Figure 44: MCTP Transmission Unit Size – NVMe Management Response**

| Bit | Description |
|---|---|
| 23:04 | Reserved |
| 15:00 | **MCTP Transmission Unit Size:** This field contains the MCTP Transmission Unit Size in bytes to be used by the port. The default value for this field following a reset or power cycle is 40h (64). |

## 5.2 Configuration Set

The Configuration Set command allows the Management Controller to modify the current configuration of a Management Endpoint.

The command uses NVMe Management Dwords 0 and 1. The format of NVMe Management Dwords 0 and 1 are shown in Figure 45 and Figure 46 respectively. There is no Request Data included in a Configuration Set command.

**Figure 45: Configuration Set – NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:08 | Configuration specific |
| 07:00 | **Configuration Identifier:** This field specifies the identifier of the Configuration that is being written. Refer to Figure 40. |

**Figure 46: Configuration Set – NVMe Management Dword 1**

| Bit | Description |
|---|---|
| 31:00 | Configuration specific |

NVMe-MI Configurations are listed in Figure 40. Specifying a reserved identifier in the Configuration Identifier field causes the command to complete with an Invalid Parameter error status.

The NVMe Management Response field is configuration specific.

### 5.2.1 SMBus/I2C Frequency (Configuration Identifier 01h)

The SMBus/I2C Frequency configuration specifies a new frequency for the SMBus port.

The configuration specific fields in NVMe Management Dword 0 are shown in Figure 47. The configuration specific fields in NVMe Management Dword 1 are reserved. NVMe Management Response field is reserved.

After successful completion of this command, the SMBus/I2C frequency is updated to the specified frequency. A Management Controller should not change this configuration while there are other Command Messages outstanding.

If the specified frequency is not supported or the Port Identifier specified is not an SMBus/I2C port, the Management Endpoint shall respond with an Invalid Parameter error status.

**Figure 47: SMBus/I2C Frequency – NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:24 | **Port Identifier:** This field specifies the port whose SMBus/I2C Frequency is specified. |
| 23:12 | Reserved |

<table>
<tr><td>10:08</td><td><b>SMBus/I2C Frequency:</b> This field specifies the new frequency for the specified SMBus/I2C port.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0h</td><td>Reserved</td></tr>
<tr><td>1h</td><td>100 kHz</td></tr>
<tr><td>2h</td><td>400 kHz</td></tr>
<tr><td>3h</td><td>1 MHz</td></tr>
<tr><td>4h - Fh</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>07:00</td><td><b>Configuration Identifier:</b> This field specifies the identifier of the Configuration that is being written. Refer to Figure 40.</td></tr>
</table>

### 5.2.2 Health Status Change (Configuration Identifier 02h)

This Configuration Identifier is used to clear selected status bits in the Composite Controller Status field of the NVM Subsystem Health Data Structure, refer to Figure 57, returned by the NVM Subsystem Health Status Poll command.

The Composite Controller Status field of the NVM Subsystem Health Data Structure is used to report the occurrence of health and status events associated with the NVM subsystem. When a bit in this field is set to ‘1’, it remains a ‘1’ until cleared.

A Configuration Set command that selects Health Status Change may be used to clear corresponding bits selected in NVMe Management Dword 1 of the Composite Controller Status field to ‘0’.

**Figure 48: Health Status Change - NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:08 | Reserved |
| 07:00 | **Configuration Identifier:** This field specifies the identifier of the Configuration that is being written. Refer to Figure 40. |

**Figure 49: Health Status Change – NVMe Management Dword 1**

| Bit | Description |
|---|---|
| 31:12 | Reserved |
| 11 | **Critical Warning:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 10 | **Available Spare:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 9 | **Percentage Used:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 8 | **Composite Temperature Change:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 7 | **Controller Status Change:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 6 | **Firmware Activated:** When this bit is set to ‘1’, the corresponding bit in the |

| | |
|---|---|
| | Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 5 | **Namespace Attribute Changed:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 4 | **Controller Enable Change Occurred:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 3 | **NVM Subsystem Reset Occurred:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 2 | **Shutdown Status:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 1 | **Controller Fatal Status:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |
| 0 | **Ready:** When this bit is set to ‘1’, the corresponding bit in the Composite Controller Status field of the NVM Subsystem Health Data Structure is cleared to ‘0’. |

### 5.2.3 MCTP Transmission Unit Size (Configuration Identifier 03h)

The MCTP Transmission Unit Size configuration specifies a new MCTP Transmission Unit Size for the specified Port Identifier. A Management Controller should check the maximum MCTP Transmission Unit Size for the port reported by the Management Endpoint using the Read NVMe-MI Data Structure command (refer to Figure 61).

The configuration specific fields in NVMe Management Dwords 0 and 1 are shown in Figure 50 and Figure 51 respectively.   The NVMe Management Response field is reserved.

After successful completion of this command, the MCTP Transmission Unit Size for MCTP packets on the specified port is updated to the specified size for future Command Messages.  A Management Controller should not change this configuration while there are other commands outstanding.

If the specified MCTP Transmission Unit Size is not supported or the Port Identifier specified is not valid, the Management Endpoint shall abort the command and send a Response Message with an Invalid Parameter error status.

**Figure 50: MCTP Transmission Unit Size – NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:24 | **Port Identifier:** This field specifies the port whose MCTP Transmission Unit Size is specified. |
| 23:08 | Reserved |
| 07:00 | **Configuration Identifier:** This field specifies the identifier of the Configuration that is being written. Refer to Figure 40. |

**Figure 51: MCTP Transmission Unit Size – NVMe Management Dword 1**

| Bit | Description |
|---|---|
| 31:16 | Reserved |

| | |
|---|---|
| 15:00 | **MCTP Transmission Unit Size:** This field contains the MCTP Transmission Unit Size in bytes to be used by the port. |

### 5.3 Controller Health Status Poll

The Controller Health Status Poll command is used to efficiently determine changes in health status attributes associated with one or more Controllers in the NVM Subsystem.

The Controller Health Status Poll command uses NVMe Management Dwords 0 and 1. The format of NVMe Management Dword 0 is shown in Figure 52 and the format of NVMe Management Dword 1 is shown in Figure 53.

**Figure 52: Controller Health Status Poll – NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31 | **Report All (ALL):** When this bit is set to '1', health status is returned for Controllers regardless of the status of the Health Status Changed flag bit vector (i.e., it is as though all the bits are set in the Health Status Changed flag bit vector). |
| 30:27 | Reserved |
| 26 | **Include SR-IOV Virtual Functions (INCVF):** When this bit is set to 1, Controller Health Status is reported for NVMe Controllers associated with SR-IOV Virtual Functions (VFs) |
| 25 | **Include SR-IOV Physical Functions (INCPF):** When this bit is set to 1, Controller Health Status is reported for NVMe Controllers associated with SR-IOV Physical Functions (PFs) |
| 24 | **Include PCI Functions (INCF):** When this bit is set to 1, Controller Health Status is reported for NVMe Controllers associated with a non SR-IOV PCI Function. |
| 23:16 | **Maximum Response Entries (MAXRENT):** This field specifies the maximum number of Controller Health Data Structure entries that may be returned in the completion. This is 0's based field. The maximum number of entries is 255. Specifying 256 entries is interpreted as an Invalid Field. |
| 15:00 | **Starting Controller ID (CTLID):** This field specifies the starting Controller ID from which to return health status information. |

**Figure 53: Controller Health Status Poll – NVMe Management Dword 1**

| Bit | Description |
|---|---|
| 31 | **Clear Changed Flags (CCF):** When this bit is set to 1, the state of reported changed flag bits in the changed flag bit vector are cleared in Controllers whose health status is contained in the Response Data. |
| 30:5 | Reserved |
| 04 | **Critical Warning (CWARN):** When this bit is set to 1, critical warning changes are reported. |
| 03 | **Available Spare (SPARE):** When this bit is set to 1, available spare changes are reported. |
| 02 | **Percentage Used (PDLU):** When this bit is set to 1, percentage used changes are reported. |
| 01 | **Composite Temperature Changes (CTEMP):** When this bit is set to 1, composite temperature changes are reported. |
| 00 | **Controller Status Changes (CSTS):** When this bit is set to 1, Controller status changes are reported. |

The Controller Health Status Poll Response Messages use the NVMe Management Response field with the format shown in Figure 54.

The Response Data field size may vary based on the number of Controllers whose health status has changed and which fields and Controller types are filtered-out. The Response Entries field indicates the number of Controller Health Data Structures that are contained in the Response Data.

**Figure 54: Controller Health Status Poll – NVMe Management Response**

| Bit | Description |
|---|---|
| 23:16 | **Response Entries (RENT):** This field specifies the number of Controller Health Data Structure Entries present in the Response Data for this Response Message. This is a 0's based value. |
| 15:00 | Reserved |

The Controller Health Data Structure, shown in Figure 55, contains the health status attributes that are tracked for each Controller. When the command is executed, health status is returned for up to 255 Controllers starting at a specified Controller ID. Controllers are ordered incrementally by Controller Identifier.

**Figure 55: Controller Health Data Structure**

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>1:0</td><td><b>Controller Identifier (CTLID):</b> This field specifies the NVMe Controller identifier with which the data contained in this data structure is associated.</td></tr>
<tr><td>3:2</td><td><b>Controller Status (CSTS):</b> This field reports the NVMe Controller status.
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>15:8</td><td>Reserved</td></tr>
<tr><td>7</td><td><b>Firmware Activated:</b> This bit is set to ‘1’ when a new firmware image is activated. This bit is cleared to ‘0’ after it is read using this command.</td></tr>
<tr><td>6</td><td><b>Namespace Attribute Changed:</b> This bit is set to ‘1’ when a change occurs in the Identify Namespace data structure for one or more namespaces. This bit is cleared to ‘0’ after it is read using this command.</td></tr>
<tr><td>5</td><td><b>Controller Enable Change Occurred:</b> This bit is set to ‘1’ when the Enable (CC.EN) bit changes state. This bit is cleared to ‘0’ after it is read using this command.</td></tr>
<tr><td>4</td><td><b>NVM Subsystem Reset Occurred:</b> This bit corresponds to the value of the NVM Subsystem Reset Occurred (CSTS.NSSRO) bit.</td></tr>
<tr><td>3:2</td><td><b>Shutdown Status:</b> This field corresponds to the value of the Shutdown Status (CSTS.SHST) field.</td></tr>
<tr><td>1</td><td><b>Controller Fatal Status:</b> This bit corresponds to the value of the Controller Fatal Status (CSTS.CFS) bit.</td></tr>
<tr><td>0</td><td><b>Ready:</b> This bit corresponds to the value of the Ready (CSTS.RDY) bit.</td></tr>
</table></td></tr>
<tr><td>5:4</td><td><b>Composite Temperature (CTEMP):</b> This field contains a value corresponding to a temperature in degrees Kelvin that represents the current composite temperature of the Controller and namespace(s) associated with that Controller. The value of this field corresponds to the value in the NVMe Controller SMART / Health Information Log.</td></tr>
<tr><td>6</td><td><b>Percentage Used (PDLU):</b> This field contains a vendor specific estimate of the percentage of NVM Subsystem life used based on the actual usage and the manufacturer’s prediction of NVM life. The value of this field corresponds to the value in the NVMe Controller SMART / Health Information Log.</td></tr>
</table>

<table>
<tr><td>7</td><td><b>Available Spare (SPARE):</b> This field contains a normalized percentage (0 to 100%) of the remaining spare capacity available. The value of this field corresponds to the value in the NVMe Controller SMART / Health Information Log.</td></tr>
<tr><td>8</td><td><b>Critical Warning (CWARN):</b> This field indicates critical warnings for the state of the Controller. The value of this field corresponds to the value in the NVMe Controller SMART / Health Information Log.
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:5</td><td>Reserved</td></tr>
<tr><td>4</td><td><b>Volatile Memory Backup Failed:</b> This bit is set to ‘1’ when the volatile memory backup device has failed.</td></tr>
<tr><td>3</td><td><b>Read Only:</b> This bit is set to ‘1’ when the media has been placed in read only mode.</td></tr>
<tr><td>2</td><td><b>Reliability Degraded:</b> This bit is set to ‘1’ when NVM Subsystem reliability has been degraded due to significant media related errors or an internal error.</td></tr>
<tr><td>1</td><td><b>Temperature Above or Under Threshold:</b> This bit is set to ‘1’ when a temperature is above an over temperature threshold or below an under temperature threshold.</td></tr>
<tr><td>0</td><td><b>Spare Threshold:</b> this bit is set to ‘1’ when the available spare has fallen below the available spare threshold.</td></tr>
</table>
</td></tr>
<tr><td>15:9</td><td>Reserved</td></tr>
</table>

Associated with each Controller in the NVM Subsystem is a Health Status Changed flag bit vector with a bit corresponding to each field in the Controller Health Data Structure. The initial value following a reset or power cycle of all changed flag bits is cleared to ‘0’. A Health Status Changed flag bit in the bit vector for a Controller is set when the value of the corresponding Controller Health Data Structure field for that Controller changes state. The state of the entire changed flag bit vector is cleared in a Controller on a reset, power cycle, or a Configuration Set command that selects Health Status Change. The state of reported bits in the changed flag bit vector is cleared in Controllers whose health status is returned in the Success Response Message to a Controller Health Status Poll command with the Clear Changed Flags bit set to ‘1’.

A Controller Health Status Poll response may return the health status for up to 255 Controllers in the Response Data field. An NVM Subsystem may contain up to 64K Controllers, so a method is needed to limit the size of the Response Message. The Starting Controller Identifier field in the Command Message specifies the starting Controller ID that is checked for a change in health status while the Maximum Response Entries field specifies the maximum number of Controllers whose health status may be returned in the Response Data field. The Response Data field contains the Controller Health Status Data Structure for the first M Controllers starting with Controller N whose health status has changed, where M is equal to the Maximum Response Entries field and N is equal to the Starting Controller Identifier field.

Health status may be filtered (i.e., excluded from being included in the Response Data field regardless of the Health Status Changed flag) by NVMe Controller type (i.e., PCI Function, SR-IOV PF, and SR-IOV VF) and by fields in the Controller Health Data Structure. NVMe Controller type filtering is selected by the Include PCI Functions, Include SR-IOV PFs, and Include SR-IOV VFs fields in NVMe Management Dword 0. When one of these bits is set, Controllers corresponding to that type of PCI Function are excluded.

Filtering of changes in individual Controller Health Data Structure fields is controlled by fields in NVMe Management Dword 1. When one of these bits is cleared to ’0’, then that field is removed from determination of health status changes for that Controller. A Controller’s health status is considered to have changed when one or more unfiltered changed flag bits in the bit vector for that Controller are set.

### 5.4 NVM Subsystem Health Status Poll

The NVM Subsystem Health Status Poll command is used to efficiently determine changes in health status attributes associated with the NVM Subsystem.

The NVM Subsystem Health Status Poll command uses NVMe Management Dword 1 as shown in Figure 56.

**Figure 56: NVM Subsystem Health Status Poll - NVMe Management Dword 1**

| Bit | Description |
|---|---|
| 31 | **Clear Status (CS):** When this bit is set to 1, the state of reported Composite Controller Status is cleared. |
| 30:0 | Reserved |

All other command specific fields are reserved.

The NVM Subsystem Health Data Structure, shown in Figure 57, is returned in the Response Data of a Successful Response Message. NVM Subsystem Health Status Poll Command responses do not use the NVMe Management Response field and this field is reserved. The Response Data field contains the NVM Subsystem Health Data Structure and is always the size of the NVM Subsystem Health Data Structure.

**Figure 57: NVM Subsystem Health Data Structure**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td><b>NVM Subsystem Status:</b> This field indicates the status of the NVM Subsystem.
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:6</td><td>Reserved</td></tr>
<tr><td>5</td><td><b>Drive Functional:</b> This bit is set to ‘1’ to indicate an NVM Subsystem is functional. If cleared to ‘0’, then there is an unrecoverable failure in the NVM Subsystem and the rest of the transmission may contain invalid information.</td></tr>
<tr><td>4</td><td><b>Reset Not Required:</b> This bit is set to ‘1’ to indicate the NVM Subsystem does not need a reset to resume normal operation. If cleared to ‘0’ then the NVM Subsystem has experienced an error that prevents continued normal operation. A Controller Level Reset is required to resume normal operation.</td></tr>
<tr><td>3</td><td><b>Port 0 PCIe Link Active:</b> This bit is set to ‘1’ to indicate the first port’s PCIe link is up (i.e., the Data Link Control and Management State Machine is in the DL_Active state). If cleared to ‘0’, then the PCIe link is down.</td></tr>
<tr><td>2</td><td><b>Port 1 PCIe Link Active:</b> This bit is set to ‘1’ to indicate the second port’s PCIe link is up. If cleared to ‘0’, then the second port’s PCIe link is down or not present.</td></tr>
<tr><td>1:0</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>1</td><td><b>Smart Warnings:</b> This field contains the Critical Warning field (byte 0) of the NVMe SMART / Health Information log. Each bit in this field is inverted from the NVMe definition (i.e., the management interface shall indicate a ‘0’ value while the corresponding bit is ‘1’ in the log page). Refer to the NVMe specification for bit definitions.<br>If there are multiple Controllers in the NVM Subsystem, the management endpoint shall combine the Critical Warning field from every Controller in the NVM Subsystem such that a bit in this field</td></tr>
</table>

<table>
<tr><td></td><td>is:<ul><li>Cleared to ‘0’ if any Controller in the subsystem indicates a critical warning for that corresponding bit.</li><li>Set to ‘1’ if all Controllers in the NVM Subsystem do not indicate a critical warning for the corresponding bit.</li></ul></td></tr>
<tr><td>2</td><td><b>Composite Temperature:</b> This field indicates the current temperature in degrees Celsius. If a temperature value is reported, it should be the same temperature as the Composite Temperature from the SMART log of hottest Controller in the NVM Subsystem. The reported temperature range is vendor specific, and shall not exceed the range -60 to +127°C. The 8 bit format of the data is shown below.<br><br>This field should not report a temperature that is older than 1 second. If recent data is not available, the Management Endpoint should indicate a value of 80h for this field.<br><br>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00h-7Eh</td><td>Temperature is measured in degrees Celsius (0 to 126C)</td></tr>
<tr><td>7Fh</td><td>127C or higher</td></tr>
<tr><td>80h</td><td>No temperature data or temperature data is more the 5 seconds old.</td></tr>
<tr><td>81h</td><td>Temperature sensor failure</td></tr>
<tr><td>82h-C3h</td><td>Reserved</td></tr>
<tr><td>C4</td><td>Temperature is -60C or lower</td></tr>
<tr><td>C5-FFh</td><td>Temperature measured in degrees Celsius is represented in two’s complement (-1 to -59C)</td></tr>
</table></td></tr>
<tr><td>3</td><td><b>Percentage Drive Life Used:</b> Contains a vendor specific estimate of the percentage of NVM Subsystem NVM life used based on the actual usage and the manufacturer’s prediction of NVM life. If an NVM Subsystem has multiple Controllers the highest value is returned. A value of 100 indicates that the estimated endurance of the NVM in the NVM Subsystem has been consumed, but may not indicate an NVM Subsystem failure. The value is allowed to exceed 100. Percentages greater than 254 shall be represented as 255. This value should be updated once per power-on hour and equal the Percentage Used value in the NVMe SMART Health Log Page.</td></tr>
<tr><td>5:4</td><td><b>Composite Controller Status:</b> This field reports the composite status of all Controllers in the NVM Subsystem.<br><br>All bits in this field are cleared to ‘0’ for a Controller during a Controller Level Reset. The bits in this field are also cleared after the NVM Subsystem Health Data Structure (refer to Figure 57) is returned in a Success Response Message associated with a NVM Subsystem Health Status Poll command where the Clear Status bit set. A Configuration Set command that selects Health Status Change may be used to clear selected bits to ‘0’.<br><br>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>15:13</td><td>Reserved</td></tr>
<tr><td>12</td><td><b>Critical Warning:</b> This bit is set to ‘1’ when the Critical Warning field in the Controller Health Data Structure is set to ‘1’ in one or more Controllers in the NVM Subsystem.</td></tr>
<tr><td>11</td><td><b>Available Spare:</b> This bit is set to ‘1’ when the Available Spare field in the Controller Health Data Structure has changed state in one or more Controllers in the NVM Subsystem</td></tr>
<tr><td>10</td><td><b>Percentage Used:</b> This bit is set to ‘1’ when the Percentage Used field in the Controller Health Data Structure is set to ‘1’ in one or more Controllers in the NVM Subsystem</td></tr>
<tr><td>9</td><td><b>Composite Temperature Change:</b> This bit is set to ‘1’ when the Composite</td></tr>
</table></td></tr>
</table>

| | | | |
|---|---|---|---|
| | | | Temperature field in the Controller Health Data Structure is set to ‘1’ in one or more Controllers in the NVM Subsystem |
| | | 8 | **Controller Status Change:** This bit is set to ‘1’ when the Controller Status field in the Controller Health Data Structure is set to ‘1’ in one or more Controllers in the NVM Subsystem |
| | | 7 | **Firmware Activated:** This bit is set to ‘1’ when a new firmware image is activated in the NVM Subsystem. |
| | | 6 | **Namespace Attribute Changed:** This bit is set to ‘1’ when a change occurred in the Identify Namespace data structure associated with one or more namespaces in the NVM Subsystem. |
| | | 5 | **Controller Enable Change Occurred:** This bit is set to ‘1’ when the Enable (CC.EN) bit changes state (i.e., ‘0’ to ‘1’ or ‘1’ to ‘0’) in one or more Controllers in the NVM Subsystem. |
| | | 4 | **NVM Subsystem Reset Occurred:** This bit is set to ‘1’ when the value of the NVM Subsystem Reset Occurred (CSTS.NSSRO) bit transitions from a ‘0’ to a ‘1’ in one or more Controllers in the NVM Subsystem. |
| | | 3:2 | **Shutdown Status:** This bit is set to ‘1’ when the value of the Shutdown Status (CSTS.SHST) field bit transitions from a ‘0’ to a ‘1’ in one or more Controllers in the NVM Subsystem. |
| | | 1 | **Controller Fatal Status:** This bit is set to ‘1’ when the value of the Controller Fatal Status (CSTS.CFS) bit transitions from a ‘0’ to a ‘1’ in one or more Controllers in the NVM Subsystem. |
| | | 0 | **Ready:** This bit is set to ‘1’ when the value of the Ready (CSTS.RDY) bit transitions from a ‘0’ to a ‘1’ in one or more Controllers in the NVM Subsystem. |
| 7:6 | Reserved | | |

### 5.5 Read NVMe-MI Data Structure

The Read NVMe-MI Data Structure command requests a data buffer that describes information about the NVM Subsystem, the Management Endpoint or the NVMe Controllers.

The command uses NVMe Management Dword 0. The format of NVMe Management Dword 0 is shown in Figure 58. NVMe Management Dword 1 is reserved. There is no Request Data included in a Read NVMe-MI Data Structure command.

**Figure 58: Read NVMe-MI Data Structure – NVMe Management Dword 0**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:24</td><td><b>Data Structure Type (DTYP):</b> This field specifies the data structure to return
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00h</td><td>NVM Subsystem Information</td></tr>
<tr><td>01h</td><td>Port Information</td></tr>
<tr><td>02h</td><td>Controller List</td></tr>
<tr><td>03h</td><td>Controller Information</td></tr>
<tr><td>04h</td><td>Optional Commands Supported</td></tr>
<tr><td>05h-FFh</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>23:16</td><td><b>Port Identifier (PORTID):</b> This field contains the identifier of the port whose data structure is returned.<br><br>If the DTYP field value corresponds to Port Information, then this field contains the port identifier whose information is requested.<br><br>For all other values of the DTYP field, this field is reserved.</td></tr>
<tr><td>15:00</td><td><b>Controller Identifier (CTRLID):</b> This field contains the Controller identifier whose data structure is returned.<br><br>If the DTYP field value corresponds to Controller List or Controller Information, then this field contains the Controller identifier in the NVM Subsystem whose information is requested.<br><br>For all other values of the DTYP field, this field is reserved.</td></tr>
</table>

Upon successful completion of the Read NVMe-MI Data Structure, the NVMe Management Response field is shown in Figure 59 and the specified data structure is returned in the Response Data.

**Figure 59: Read NVMe-MI Data Structure – NVMe Management Response**

| Bit | Description |
|---|---|
| 23:16 | Reserved |
| 15:00 | **Response Data Length:** The length, in bytes, of the Response Data field in this Response Message. |

The NVM Subsystem Information data structure contains information about the NVM Subsystem. The Port Identifier and Controller Identifier fields are reserved. The format is shown in Figure 60.

**Figure 60: NVM Subsystem Information Data Structure**

| Byte | Description |
|---|---|
| 00 | **Number of Ports (NUMP):** This field specifies the maximum number of ports of any type supported by the NVM Subsystem. This is a 0’s based value. |
| 01 | **NVMe-MI Major Version Number (MJR):** This field shall be set to 1h to indicate the major version number of this specification. |
| 02 | **NVMe-MI Minor Version Number (MNR):** This field shall be cleared to 0h to indicate the minor version number of this specification. |
| 31:03 | Reserved |

The Port Information data structure contains information about a port within the NVM Subsystem. The Port Identifier specifies the port. The Controller Identifier fields are reserved. The format is shown in Figure 61.

**Figure 61: Port Information Data Structure**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td><b>Port Type:</b> Specifies the port type.<br>
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0h</td><td>Inactive</td></tr>
<tr><td>1h</td><td>PCIe</td></tr>
<tr><td>2h</td><td>SMBus</td></tr>
<tr><td>3h – FFh</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>01</td><td>Reserved</td></tr>
<tr><td>03:02</td><td><b>Maximum MCTP Transmission Unit Size:</b> The maximum MCTP Transmission Unit size the port is capable of sending and receiving.<br>If the port does not support MCTP, then this field shall be set to 0.<br>If the port type is PCIe and the port supports MCTP, then this field shall be set to a value between 64 bytes and the PCIe Max Payload Size supported minus 4, inclusive. All PCIe ports within an NVM Subsystem should report the same value in this field.<br>If the port type is SMBus and the port supports MCTP, then this field shall be set to a value between 64 bytes and 250 bytes, inclusive.</td></tr>
<tr><td>07:04</td><td>Reserved</td></tr>
<tr><td>31:08</td><td>Port Type Specific (refer to Figure 62 and Figure 63)</td></tr>
</table>

**Figure 62: PCIe Port Specific Data**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>08</td><td><b>PCIe Maximum Payload Size:</b> This field indicates the Max Payload Size for the specified PCIe port. If the link is not active, this field should be cleared to 0h.<br>
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0h</td><td>128 bytes</td></tr>
<tr><td>1h</td><td>256 bytes</td></tr>
<tr><td>2h</td><td>512 bytes</td></tr>
<tr><td>3h</td><td>1024 bytes</td></tr>
<tr><td>4h</td><td>2048 bytes</td></tr>
<tr><td>5h</td><td>4096 bytes</td></tr>
<tr><td>6h-FFh</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>09</td><td><b>PCIe Supported Link Speeds Vector:</b> This field indicates the Supported Link Speeds for the specified PCIe port.<br>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:3</td><td>Reserved</td></tr>
<tr><td>2</td><td>This bit shall be set to ‘1’ if the link supports 8.0 GT/s</td></tr>
<tr><td>1</td><td>This bit shall be set to ‘1’ if the link supports 5.0 GT/s</td></tr>
<tr><td>0</td><td>This bit shall be set to ‘1’ if the link supports 2.5 GT/s.</td></tr>
</table></td></tr>
<tr><td>10</td><td><b>PCIe Current Link Speed:</b> The port’s PCIe negotiated link speed using the same encoding as</td></tr>
</table>

<table>
<tr>
<td></td>
<td>the PCIe Supported Link Speed Vector field.  A value of 0h in this field indicates the PCIe Link is not available.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0h</td><td>Link not active</td></tr>
<tr><td>1h</td><td>The current link speed is the speed indicated in the supported link speed bit 0.</td></tr>
<tr><td>2h</td><td>The current link speed is the speed indicated in the supported link speed bit 1.</td></tr>
<tr><td>3h</td><td>The current link speed is the speed indicated in the supported link speed bit 2.</td></tr>
<tr><td>4h</td><td>The current link speed is the speed indicated in the supported link speed bit 3.</td></tr>
<tr><td>5h</td><td>The current link speed is the speed indicated in the supported link speed bit 4.</td></tr>
<tr><td>6h</td><td>The current link speed is the speed indicated in the supported link speed bit 5.</td></tr>
<tr><td>7h</td><td>The current link speed is the speed indicated in the supported link speed bit 6.</td></tr>
<tr><td>8h-FFh</td><td>Reserved</td></tr>
</table>
</td>
</tr>
<tr>
<td>11</td>
<td><b>PCIe Maximum Link Width:</b> The maximum PCIe link width for this NVM Subsystem port.  This is the expected negotiated link width that the port link trains to if the platform supports it.  A Management Controller may compare this value with the PCIe Negotiated Link Width to determine if there has been a PCIe link training issue.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>PCIe x1</td></tr>
<tr><td>2</td><td>PCIe x2</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
<tr><td>4</td><td>PCIe x4</td></tr>
<tr><td>5-7</td><td>Reserved</td></tr>
<tr><td>8</td><td>PCIe x8</td></tr>
<tr><td>9-11</td><td>Reserved</td></tr>
<tr><td>12</td><td>PCIe x12</td></tr>
<tr><td>13-15</td><td>Reserved</td></tr>
<tr><td>16</td><td>PCIe x16</td></tr>
<tr><td>17-31</td><td>Reserved</td></tr>
<tr><td>32</td><td>PCIe x32</td></tr>
<tr><td>33-255</td><td>Reserved</td></tr>
</table>
</td>
</tr>
<tr>
<td>12</td>
<td><b>PCIe Negotiated Link Width:</b> The negotiated PCIe link width for this port.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0</td><td>Link not active</td></tr>
<tr><td>1</td><td>PCIe x1</td></tr>
<tr><td>2</td><td>PCIe x2</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
<tr><td>4</td><td>PCIe x4</td></tr>
<tr><td>5-7</td><td>Reserved</td></tr>
<tr><td>8</td><td>PCIe x8</td></tr>
<tr><td>9-11</td><td>Reserved</td></tr>
<tr><td>12</td><td>PCIe x12</td></tr>
<tr><td>13-15</td><td>Reserved</td></tr>
<tr><td>16</td><td>PCIe x16</td></tr>
<tr><td>17-31</td><td>Reserved</td></tr>
<tr><td>32</td><td>PCIe x32</td></tr>
<tr><td>33-255</td><td>Reserved</td></tr>
</table>
</td>
</tr>
</table>

| | |
|---|---|
| 31:13 | Reserved |

**Figure 63: SMBus Port Specific Data**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>08</td><td><b>Current VPD SMBus/I2C Address:</b> This field indicates the current VPD SMBus/I2C address.  A value of 0h indicates there is no VPD.</td></tr>
<tr><td>09</td><td><b>Maximum VPD Access SMBus/I2C Frequency:</b> This field indicates the maximum SMBus/I2C frequency supported on the VPD interface.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0h</td><td>Not supported</td></tr>
<tr><td>1h</td><td>100 kHz</td></tr>
<tr><td>2h</td><td>400 kHz</td></tr>
<tr><td>3h</td><td>1 MHz</td></tr>
<tr><td>4-FFh</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>10</td><td><b>Current Management Endpoint SMBus/I2C Address:</b> This field indicates the current MCTP SMBus/I2C address. A value of 0h indicates there is no Management Endpoint on this port.</td></tr>
<tr><td>11</td><td><b>Maximum Management Endpoint SMBus/I2C Frequency:</b> This field indicates the maximum SMBus/I2C frequency supported by the Management Endpoint.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0h</td><td>Not supported</td></tr>
<tr><td>1h</td><td>100 kHz</td></tr>
<tr><td>2h</td><td>400 kHz</td></tr>
<tr><td>3h</td><td>1 MHz</td></tr>
<tr><td>4-FFh</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>12</td><td><b>NVMe Basic Management:</b> Bit 0 in this field, if set to ‘1’, indicates if the port implements the NVMe Basic Management command specified in Appendix A.  All other bits in this field are reserved.</td></tr>
<tr><td>31:13</td><td>Reserved</td></tr>
</table>

The Controller List data structure contains a list of NVMe Controllers in the NVM Subsystem greater than or equal to the value specified in the Controller Identifier (CTRLID) field.  A Controller List may contain up to 2047 Controller identifiers.  Refer to the NVM Express specification for a definition of the Controller List data structure.

**Figure 64: Controller Information Data Structure**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td><b>Port Identifier (PORTID):</b> This field specifies the PCIe port identifier with which the Controller is associated.</td></tr>
<tr><td>04:01</td><td>Reserved</td></tr>
<tr><td>05</td><td><b>PCIe Routing ID Information (PRII):</b>  This field provides additional data about the PCI Express Routing ID (PRI) for the specified Controller.
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:1</td><td>Reserved</td></tr>
<tr><td>0</td><td><b>PCIe Routing ID Valid:</b> This bit is set to ‘1’ if the device has captured a Bus Number and Device Number (Bus Number only for ARI devices).  This bit is set to ‘0’ if the device has not captured a Bus and Device number (Bus Number only for ARI devices).</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>07:06</td><td><b>PCIe Routing ID (PRI):</b> This field contains the PCIe Routing ID for the specified Controller.
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>15:8</td><td><b>PCI Bus Number:</b> The Controller’s PCI Bus Number.</td></tr>
<tr><td>7:3</td><td><b>PCI Device Number:</b> The Controller’s PCI Device Number.</td></tr>
<tr><td>2:0</td><td><b>PCI Function Number:</b> The Controller’s PCI Function Number.</td></tr>
</table>
Note: For an ARI Device, bits 7:0 represents the (8-bit) Function Number, which replaces the (5-bit) Device Number and (3-bit) Function Number fields above.</td></tr>
<tr><td>09:08</td><td><b>PCI Vendor ID:</b> The PCI Vendor ID for the specified Controller.</td></tr>
<tr><td>11:10</td><td><b>PCI Device ID:</b> The PCI Device ID for the specified Controller.</td></tr>
<tr><td>13:12</td><td><b>PCI Subsystem Vendor ID:</b> The PCI Subsystem Vendor ID for the specified Controller.</td></tr>
<tr><td>15:14</td><td><b>PCI Subsystem Device ID:</b> The PCI Subsystem Device ID for the specified Controller.</td></tr>
<tr><td>31:16</td><td>Reserved</td></tr>
</table>

The Optionally Supported Command List data structure contains a list of optional commands that a Management Endpoint supports. The Optionally Supported Command List data structure may contain up to 2047 commands, and shall be minimally sized (i.e., if there is 1 optionally supported command, the data structure is 4 bytes total).

**Figure 65: Optionally Supported Command List Data Structure**

| Byte | Description |
|---|---|
| 01:00 | **Number of Commands (NUMCMD):** This field contains the number of optionally supported commands in the list. A value of 0h indicates there are no commands in the list. |
| 03:02 | **Command 0 (CMD0):** This field contains the Command Type and Opcode for the first optionally supported command or 0h if the list is empty (i.e. no optional commands are supported). Refer to Figure 66. |
| 05:04 | **Command 1 (CMD1):** This field contains the Command Type and Opcode for the second optionally supported command, if applicable. Refer to Figure 66. |
| … | |
| (N*2 +3):<br>(N*2 + 2) | **Command N (CMDN):** This field contains the Command Type and Opcode for the N+1 optionally supported command, if applicable. Refer to Figure 66. |

**Figure 66: Optionally Supported Command Data Structure**

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td><b>Command Type:</b> This field specifies the command set used by the optionally supported command.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved</td></tr>
<tr><td>6:3</td><td><b>NVMe-MI Message Type (NMIMT):</b> This field specifies the NVMe-MI Message Type. Refer to Figure 11.</td></tr>
<tr><td>2:0</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>01</td><td><b>Opcode:</b> This field specifies the opcode used for the optionally supported command.</td></tr>
</table>

### 5.6 Reset

The Reset command may be used to initiate a reset.

The Reset command uses NVMe management Dword 0. The format of NVMe Management Dword 0 is shown in Figure 67. All other command specific fields in the Request Message and Response Message are reserved.

**Figure 67: Reset - NVMe Management Dword 0**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:24</td><td><b>Reset Type:</b> This field specifies the type of reset to be performed.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00h</td><td>Reset NVM Subsystem</td></tr>
<tr><td>01h – FFh</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>23:00</td><td>Reserved</td></tr>
</table>

When a Reset command is completed successfully, the NVM Subsystem Reset is immediately initiated (refer to 9.3). No success response is transmitted.

### 5.7 VPD Read

The VPD Read command is used to read the Vital Product Data described in section 9.2. Upon successful completion of the VPD Read command, the specified portion of the VPD contents is returned in the Response Data.

The VPD Read command uses NVMe Management Dword 0 and 1. The format of NVMe Management Dwords 0 and 1 are shown in Figure 68 and Figure 69 respectively. There is no Request Data sent in the Request Message.

A VPD Read command with length 0 and no data is valid. The Management Endpoint responds with a Success Response Message and no Response Data. If the Data Length plus Data Offset fields are greater than the size of the VPD, then the Management Endpoint does not return the VPD contents and responds with an Invalid Parameter error status response.

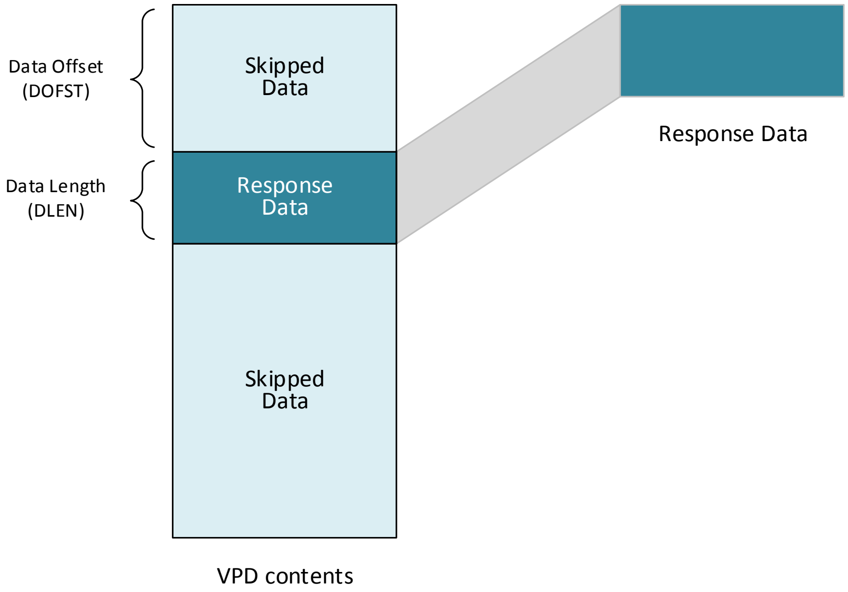
**Figure 68: VPD Read NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:16 | Reserved |
| 15:00 | **Data Offset (DOFST): T**his field specifies the starting offset, in bytes, into the VPD data that is contained in the Response Message. |

**Figure 69: VPD Read NVMe Management Dword 1**

| Bit | Description |
|---|---|
| 31:16 | Reserved |
| 15:00 | **Data Length (DLEN):** This field specifies the length, in bytes, to be read from the VPD starting at the byte offset specified by DOFST. |

**Figure 70: VPD Read Response Data**



### 5.8 VPD Write

The VPD Write command is used to update the Vital Product Data described in section 9.2.

After the VPD Write command completes successfully, reading the contents of the FRU Information Device directly or executing a VPD Read command shall return the new VPD contents (i.e., those supplied with the VPD Write command). The data to be written to the VPD is specified in the Request Data field. VPD Write uses NVMe Management Dwords 0 and 1 as shown in Figure 71 and Figure 72.

The VPD contents should be capable of being updated at least 100 times using the VPD Write command. If there is an error preventing update of the VPD contents, then the Management Endpoint responds with a generic error response and VPD Writes Exceeded status.

A VPD Write command with length 0 and no data is valid. The Management Endpoint responds with a Success Response Message.

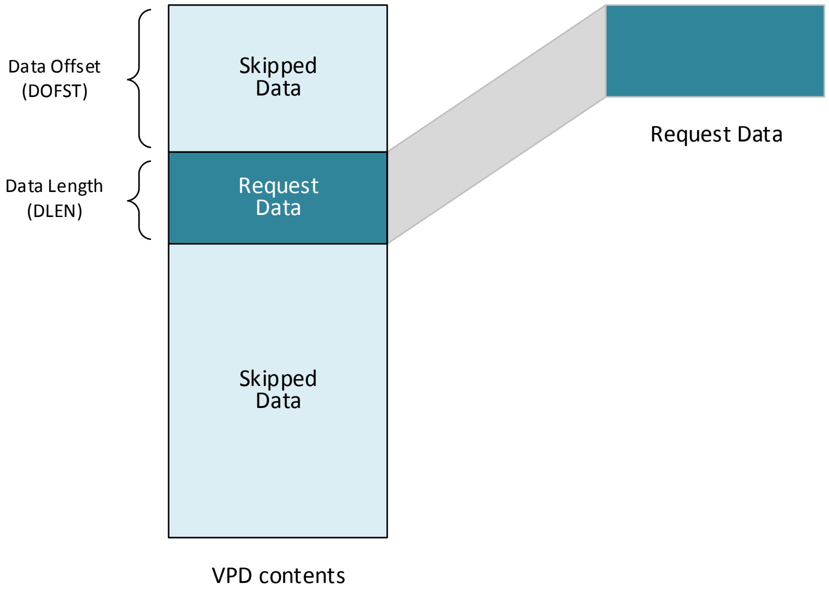
**Figure 71: VPD Write – NVMe Management Dword 0**

| Bit | Description |
|---|---|
| 31:16 | Reserved |
| 15:00 | **Data Offset (DOFST): T**his field specifies the starting offset, in bytes, into the VPD data that is written. |

**Figure 72: VPD Write – NVMe Management Dword 1**

| Bit | Description |
|---|---|
| 31:16 | Reserved |
| 15:00 | **Data Length (DLEN):** This field specifies the length, in bytes, to be written to the VPD starting at the byte offset specified by DOFST. |

**Figure 73: VPD Write Request Data**



The Management Controller should not read the contents of the VPD while this command is processing. Reading the contents of the VPD or executing a VPD Read command while a VPD Write command is executing may return incorrect data as a result of the read.

If the Data Length plus Data Offset fields are greater than the size of the VPD, then the Management Endpoint does not write to the VPD and responds with an Invalid Parameter error status response.

# 6 NVM Express Admin Command Set

The NVM Express Admin Command Set allows NVMe Admin commands to be issued to any Controller in the NVM Subsystem using NVMe-MI. Supported commands are listed in Figure 74, and are defined in the NVMe specification. If an NVMe Admin Command is issued in a Request Message other than one listed in Figure 74, the Management Endpoint shall return a response with status Invalid Parameter pointing to the NVMe opcode. Future revisions of this specification may add additional commands to Figure 74.

**Figure 74: List of NVMe Admin Commands Supported**

| Command | O/M[1] |
|---|---|
| Firmware Activate/Commit | O |
| Firmware Image Download | O |
| Format NVM | O |
| Get Features | M |
| Get Log Page | M |
| Identify | M |
| Namespace Management | O |
| Namespace Attachment | O |
| Security Send | O |
| Security Receive | O |
| Set Features | O |
| Vendor Specific | O |

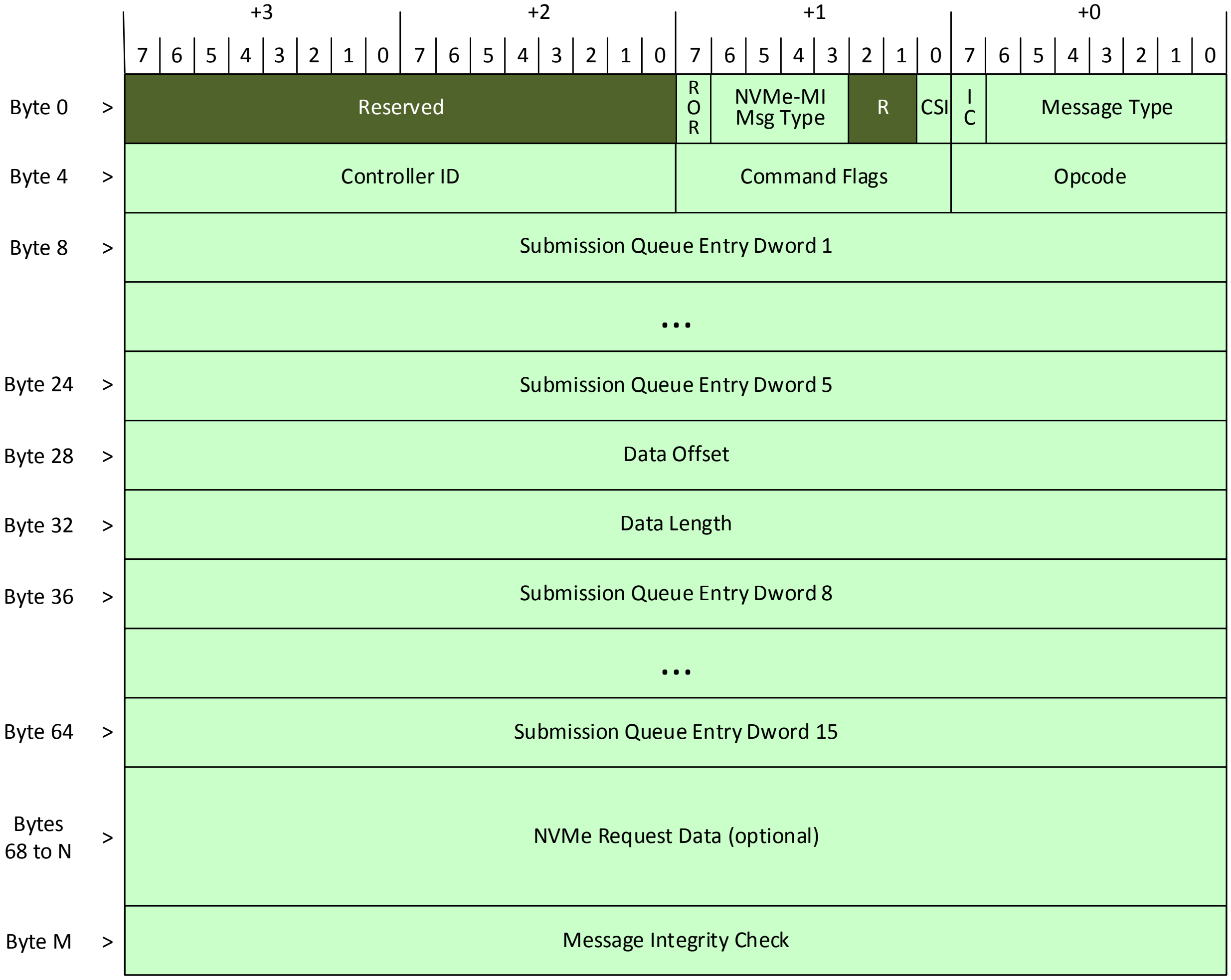
NOTES:
1. O/M definition: O = Optional, M = Mandatory. Mandatory commands shall be supportd if the NVMe Controller specified by the Controller ID field supports the command.

NVMe Admin commands over NVMe-MI may interfere with host software. A Management Controller should coordinate with the host or issue only NVMe Admin commands that do not interfere with host software or in band NVMe commands (e.g., Identify). Coordination between a Management Controller and host is outside the scope of this specification.

NVMe Admin Commands over NVMe-MI may target a controller that is disabled or held in reset by the host. When this occurs, the NVMe Admin command is processed normally.

The Request Message format for NVMe Admin Commands is shown in Figure 75 and is described in Figure 76.

**Figure 75: NVMe Admin Command Request Format**



| | +3 | +2 | +1 | +0 |
|---|---|---|---|---|
| | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
| Byte 0 > | Reserved | | ROR<br>NVMe-MI Msg Type<br>R<br>CSI | IC<br>Message Type |
| Byte 4 > | Controller ID | | Command Flags | Opcode |
| Byte 8 > | Submission Queue Entry Dword 1 | | | |
| | ••• | | | |
| Byte 24 > | Submission Queue Entry Dword 5 | | | |
| Byte 28 > | Data Offset | | | |
| Byte 32 > | Data Length | | | |
| Byte 36 > | Submission Queue Entry Dword 8 | | | |
| | ••• | | | |
| Byte 64 > | Submission Queue Entry Dword 15 | | | |
| Bytes 68 to N > | NVMe Request Data (optional) | | | |
| Byte M > | Message Integrity Check | | | |

**Figure 76: NVMe Admin Command Request Description**

| Byte | Description |
|---|---|
| 03:00 | **NVMe-MI Message Header**: Refer to 3.2. |
| 04 | **Opcode (OPC):** This field specifies the opcode of the command to be executed. Refer to the NVMe specification. |
| 05 | **Command Flags (CFLGS):** This field specifies flags for the command.<br>Bits 2-7 are reserved.<br>Bit 1, if set to ‘1’ then the command contains an offset value in bytes 28-31. If cleared to ‘0’ then the DOFST field shall be cleared to 0h.<br>Bit 0, if set to ‘1’ then the command contains a length value in bytes 32-35. If cleared to ‘0’ then the DLEN field shall be cleared to 0h. |

| | |
|---|---|
| 07:06 | **Controller ID (CTLID):** This field specifies the conntroller ID of the NVMe Controller that this command targets. |
| 11:08 | **Submission Queue Entry Dword 1 (SQEDW1):** Submission Queue Entry Dword 1 as defined in the NVMe specification |
| 15:12 | **Submission Queue Entry Dword 2 (SQEDW2):** Submission Queue Entry Dword 2 as defined in the NVMe specification |
| 19:16 | **Submission Queue Entry Dword 3 (SQEDW3):** Submission Queue Entry Dword 3 as defined in the NVMe specification |
| 23:20 | **Submission Queue Entry Dword 4 (SQEDW4):** Submission Queue Entry Dword 4 as defined in the NVMe specification |
| 27:24 | **Submission Queue Entry Dword 5 (SQEDW5):** Submission Queue Entry Dword 5 as defined in the NVMe specification |
| 31:28 | **Data Offset (DOFST):** For commands that transmit data from the Management Controller to the Management Endpoint (i.e., the NVMe Data field in the Request Message has non-zero length) or do not transmit data, this field shall be cleared to ‘0’. If this field is not 0h, then the Management Endpoint shall return an error response with status Invalid Parameter.<br>For commands that transmit data from the Management Endpoint to the Management Controller (i.e., the NVMe Data field in the Response Message has non-zero length), this field specifies the starting offset, in bytes, into the completion data contained in the Response Message.<br>Bits 0 and 1 of this field shall be cleared to ‘0’. |
| 35:32 | **Data Length (DLEN):** For commands that do not transmit data in neither the Request Message nor Response Message, this field shall be cleared to 0h. If this field is not 0h, then the Management Endpoint shall return an error response with status Invalid Parameter.<br>For commands that transmit data from the Management Controller to the Management Endpoint (i.e., the NVMe Data field in the Request Message has non-zero length), this field specifies the length, in bytes, of the data contained in the Request Message.<br>For commands that transmit data from the Management Endpoint to the Management Controller (i.e., the NVMe Data field in the Response Message has non-zero length), this field specifies the length, in bytes, of the data contained in the Response Message.<br>Bits 0 and 1 of this field shall be cleared to ‘0’. This field shall be less than or equal to 4096. |
| 39:36 | **Submission Queue Entry Dword 8 (SQEDW8):** Submission Queue Entry Dword 8 as defined in the NVMe specification |
| 43:40 | **Submission Queue Entry Dword 9 (SQEDW9):** Submission Queue Entry Dword 9 as defined in the NVMe specification |
| 4744 | **Submission Queue Entry Dword 10 (SQEDW10):** Submission Queue Entry Dword 10 as defined in the NVMe specification |
| 51:48 | **Submission Queue Entry Dword 11 (SQEDW11):** Submission Queue Entry Dword 11 as defined in the NVMe specification |
| 55:52 | **Submission Queue Entry Dword 12 (SQEDW12):** Submission Queue Entry Dword 12 as defined in the NVMe specification |
| 59:56 | **Submission Queue Entry Dword 13 (SQEDW13):** Submission Queue Entry Dword 13 as defined in the NVMe specification |
| 63:60 | **Submission Queue Entry Dword 14 (SQEDW14):** Submission Queue Entry Dword 14 as defined in the NVMe specification |

| | |
|---|---|
| 67:64 | **Submission Queue Entry Dword 15 (SQEDW15):** Submission Queue Entry Dword 15 as defined in the NVMe specification |
| N:68 | **NVMe Request Data (Optional)** |
| M+3:M | **Message Integrity Check (MIC):** Refer to 3.2. |

The Response Message contains the corresponding format for NVMe Admin Commands is shown in Figure 77 and is described in Figure 78.

**Figure 77: NVMe Admin Command Response Format**

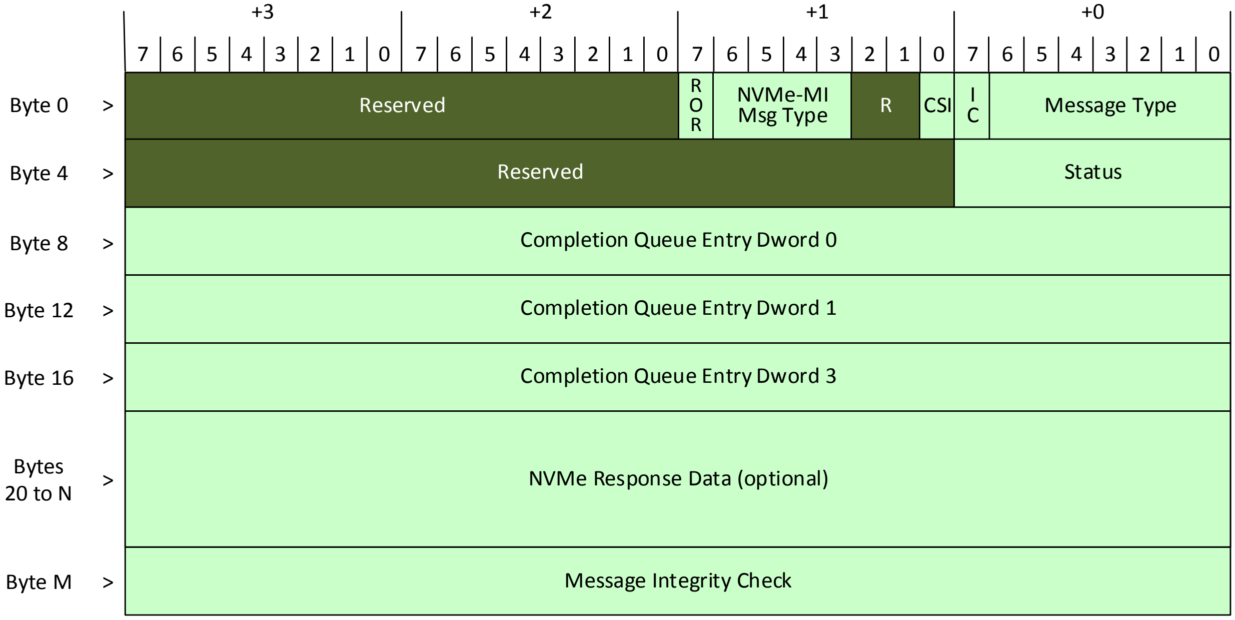


**Figure 78: NVMe Admin Command Response Description**

| Byte | Description |
|---|---|
| 03:00 | **NVMe-MI Message Header**: Refer to 3.2. |
| 04 | **Status:** This field indicates the status of the NVMe-MI command.  Refer to 4.2. |
| 07:05 | Reserved |
| 11:08 | **Completion Queue Entry Dword 0 (CQEDW0):** Completion Queue Entry Dword 0 as defined in the NVMe specification |
| 15:12 | **Completion Queue Entry Dword 1 (CQEDW1):** Completion Queue Entry Dword 1 as defined in the NVMe specification |
| 19:16 | **Completion Queue Entry Dword 3 (CQEDW3):** Completion Queue Entry Dword 3 as defined in the NVMe specification.  The Command ID field shall be cleared to 0h. |
| N:20 | **NVMe Response Data** (optional) |
| M+3:M | **Message Integrity Check**: Refer to 3.2. |

## 6.1 Request and Response Data

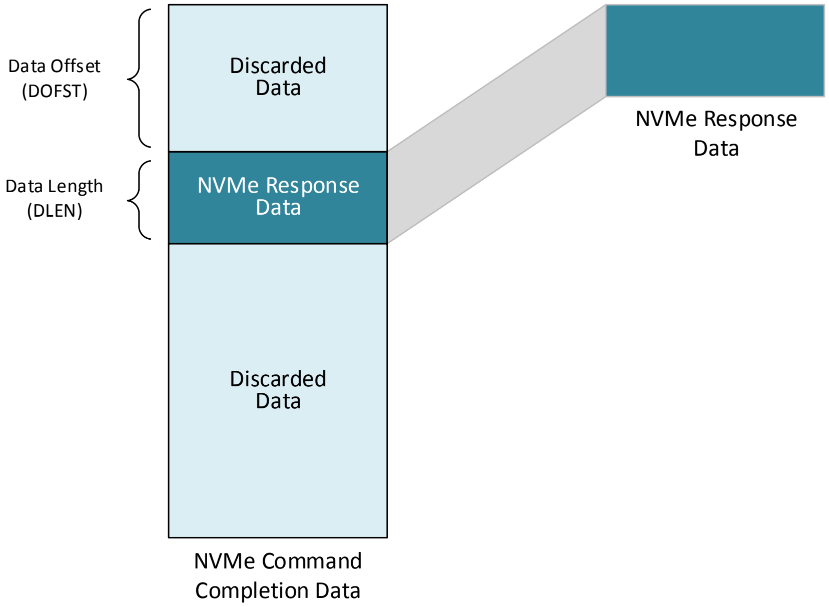
NVMe Admin Commands may contain data as part of the Command Message. This data is passed in the NVMe Data field instead of using PRP Lists or SGL segments.

If there is no data sent with the NVMe Admin Command (i.e., the Data Transfer subfield for the opcode is 00b), then the Data Offset and Data Length fields shall be cleared to 0h.

If there is data sent with the NVMe Admin Command (i.e., the Data Transfer subfield for the opcode is 01b), then the Data Offset field shall be 0h and the Data Length field shall be set to the length of the input data required by the command. If the Data Length field does not correspond to the required length, the Management Endpoint shall respond with an Invalid Parameter error status response.

If there is data expected in the Response Message in the completion of the NVMe Admin Command (i.e., the Data Transfer subfield in the corresponding NVMe Admin Command for the opcode is 10b), then the Data Offset and Data Length fields describe the portion of the completion data that is transferred in the Response Message. Any remaining data not transferred in the Response Message is discarded by the Management Endpoint as shown in Figure 79. If the Data Length plus Data Offset fields are greater than the size of the NVMe command completion data, the Management Endpoint should respond with an Invalid Parameter error status response.

**Figure 79: NVMe Admin Command Response Data Example**



## 6.2 Status

A Response Message for an NVMe Admin Command may contain two status fields. The first status field, contained in Byte 4 of the Response Message, is defined by this specification, and the second Status Field, if present, is contained in Completion Queue Entry Dword 3 and defined in the NVMe Specification.

An NVMe Admin Command Request Message is well formed if it does not contain one of the following errors:

- Invalid Opcode (e.g., the opcode is not listed in Figure 74)
- Invalid Parameter (e.g., the Controller ID field specifies a Controller ID not implemented in the NVM Subsystem)
- Invalid Command Size (e.g., the Request Message does not contain a complete command)

- Invalid Command Input Data Size (e.g., the NVMe Request Data field is larger than the size specified in the Data Length field)

If the NVMe Admin Command Request Message is well formed, then a success Response Message is transmitted.  The success response contains the status associated with NVMe Admin Command in the Status Field of Completion Queue Entry Dword 3.  The Status Field contains any NVMe specific status codes (e.g., Success or Invalid Field in Command).

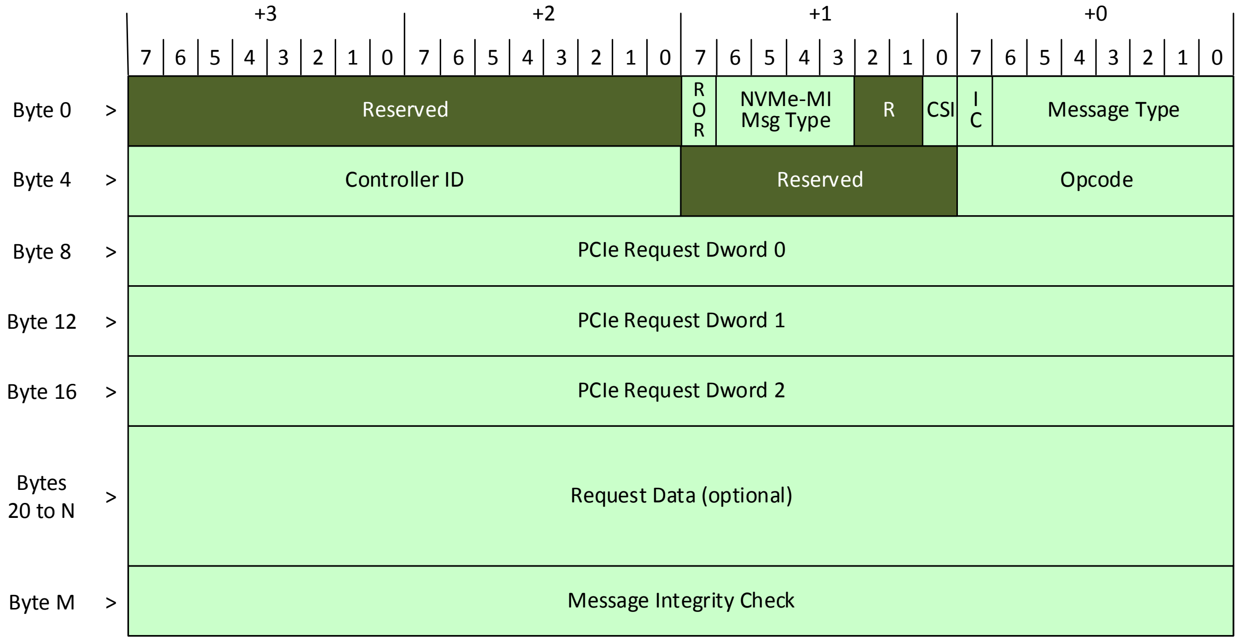
# 7 PCIe Command Set (optional)

The PCIe Command Set defines optional commands that a Management Controller may submit to access the memory, I/O, and configuration addresses spaces associated with a Controller in the NVM Subsystem. Only addresses mapped to the specified Controller may be accessed (e.g., these commands do not directly access memory on a host). The NMIMT field in the message header for PCIe Command Messages and Response Messages is set to 4h (PCIe Command).

PCIe commands over NVMe-MI may interfere with host software. A Management Controller should coordinate with the host or issue only PCIe commands that do not interfere with host software or in-band NVMe commands (e.g., PCIe Configuration Read). Coordination between a Management Controller and a host is outside the scope of this specification.

The Request Message format for PCIe Commands is shown in Figure 80 and described in Figure 81.

**Figure 80: PCIe Command Request Format**



<table>
<tr><th></th><th></th><th colspan="8">+3</th><th colspan="8">+2</th><th colspan="8">+1</th><th colspan="8">+0</th></tr>
<tr><td></td><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>></td><td colspan="16">Reserved</td><td>ROR</td><td colspan="4">NVMe-MI Msg Type</td><td colspan="2">R</td><td>CSI</td><td>IC</td><td colspan="7">Message Type</td></tr>
<tr><td>Byte 4</td><td>></td><td colspan="16">Controller ID</td><td colspan="8">Reserved</td><td colspan="8">Opcode</td></tr>
<tr><td>Byte 8</td><td>></td><td colspan="32">PCIe Request Dword 0</td></tr>
<tr><td>Byte 12</td><td>></td><td colspan="32">PCIe Request Dword 1</td></tr>
<tr><td>Byte 16</td><td>></td><td colspan="32">PCIe Request Dword 2</td></tr>
<tr><td>Bytes 20 to N</td><td>></td><td colspan="32">Request Data (optional)</td></tr>
<tr><td>Byte M</td><td>></td><td colspan="32">Message Integrity Check</td></tr>
</table>

**Figure 81: PCIe Command Request Description**

| Byte | Description |
|---|---|
| 03:00 | **NVMe-MI Message Header:** Refer to 3.2. |
| 04 | **Opcode (OPC):** This field specifies the opcode of the command to be executed. Refer to Figure 82. |
| 05 | Reserved |
| 07:06 | **Controller ID (CTLID):** This field specifies the Controller ID of the NVMe Controller that this command targets. |
| 11:08 | **PCIe Request Dword 0 (NMD0):** This field is command specific Dword 0. |
| 15:12 | **PCIe Request Dword 1 (NMD1):** This field is command specific Dword 1. |
| 19:16 | **PCIe Request Dword 2 (NMD2):** This field is command specific Dword 2. |
| N:20 | **Request Data (Optional)** |
| M+3:M | **Message Integrity Check (MIC):** Refer to 3.2. |

Figure 82 defines the PCIe Command opcodes.

**Figure 82: Opcodes for PCIe Commands**

| Opcode | O/M[1] | Command |
|---|---|---|
| 00h | O | PCIe Configuration Read |
| 01h | O | PCIe Configuration Write |
| 02h | O | PCIe Memory Read |
| 03h | O | PCIe Memory Write |
| 04h | O | PCIe I/O Read |
| 05h | O | PCIe I/O Write |
| 06h – FFh | - | Reserved |
| NOTES:<br>1. O/M definition: O = Optional, M = Mandatory. | | |

The Response Message for PCIe Command is shown in Figure 83 and described in Figure 84.

**Figure 83: PCIe Command Response Format**

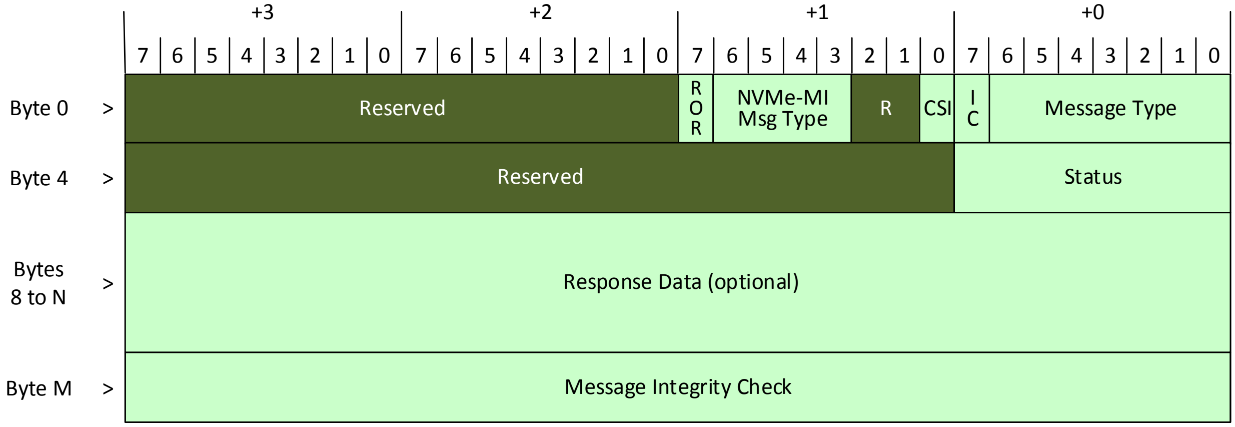


**Figure 84: PCIe Command Response Description**

| Byte | Description |
|---|---|
| 03:00 | **NVMe-MI Message Header**: Refer to 3.2. |
| 04 | **Status:** This field indicates the status of the NVMe-MI command.  Refer to 4.2. |
| 07:05 | Reserved |
| N:08 | **Response Data** (optional) |
| M+3:M | **Message Integrity Check:** Refer to 3.2. |

PCIe commands allow the Management Controller to access PCI Express configuration, I/O, and memory spaces of any Controller in the NVM Subsystem. Support for PCIe commands is optional and indicated by the Optionally Supported Commands data structure.  Refer to Figure 65.

An implementation may support a subset of the PCIe commands.  For supported commands, an implementation may block access to certain address space ranges (e.g., due to security concerns). A PCIe Command that attempts to access such a blocked address range is aborted with the Status field set to Access Denied.

It is recommended that PCIe Commands provide access to all non-blocked address spaces whenever MCTP access is supported. In some implementations, it may not be possible to access PCIe resources in certain states.  A PCIe Command executed when a Controller is in one of these states may be aborted with the Status field set to PCIe Inaccessible.  Refer to 9.1.

A PCIe command that is not well formed results in an error response. A PCIe command is well formed if it does not contain one of the following errors:

- Invalid Opcode (e.g., the Opcode is not listed in Figure 82)
- Invalid Parameter (e.g., the Controller ID field specifies a Controller ID not implemented in the NVM Subsystem)
- Invalid Command Size (e.g., the Request Message does not contain a complete command)
- Invalid Command Input Data Size (e.g., the NVMe Request Data field is larger than the size expected by the command)

## 7.1 PCIe Configuration Read

The PCIe Configuration Read command allows the Management Controller to read the contents of the PCIe configuration address space associated with an NVMe Controller in the NVM Subsystem. The Controller ID field in the Command Message specifies the Controller ID that is being accessed.

The command uses PCIe Request Dwords 0 and 1. PCIe Request Dword 2 is not used and is reserved. The format of PCIe Request Dwords 0 and 1 are shown in Figure 85 and Figure 86 respectively.

**Figure 85: PCIe Configuration Read – PCIe Request Dword 0**

| Bit | Description |
|---|---|
| 31:16 | Reserved |
| 15:00 | **Length (LENGTH):** This field specifies the number of bytes to be read. |

**Figure 86: PCIe Configuration Read – PCIe Request Dword 1**

| Bit | Description |
|---|---|
| 31:12 | Reserved |
| 11:00 | **Offset (OFFSET):** This field specifies the offset in bytes into the 4096B configuration space associated with the NVMe Controller at which the read begins. |

When this command is completed successfully, PCI configuration space associated with the NVMe Controller specified by Controller ID is read and returned in the Response Data field. The Offset field specifies the starting read offset in PCIe configuration address space and the Length field specifies the number of bytes to be read. The Response Data field is always an integral number of Dwords and is equal to the Length field rounded up to the next Dword. If Length is not an integral number of Dwords, then zero padding follows read data.

If the sum of the Offset and Length fields fall outside of PCI configuration space, then the Management Endpoint responds with an Invalid Parameter error status response. The parameter with the error in this case is always the Offset field.

A Management Endpoint shall support the PCIe Configuration Read command if any of the other PCIe Command Set commands are supported. Access to the BAR offsets shall not return an Access Denied Response Message Status code (i.e., the correct data shall be provided).

## 7.2 PCIe Configuration Write

The PCIe Configuration Write command allows the Management Controller to write the contents of the PCIe configuration address space associated with an NVMe Controller in the NVM Subsystem. The Controller ID field in the Command Message specifies the Controller ID that is being accessed.

The command uses PCIe Request Dwords 0 and 1. PCIe Request Dword 2 is not used and is reserved. The format of PCIe Request Dwords 0 and 1 are shown in Figure 87 and Figure 88 respectively.

**Figure 87: PCIe Configuration Write – PCIe Request Dword 0**

| Bit | Description |
|---|---|
| 31:16 | Reserved |
| 15:00 | **Length (LENGTH):** This field specifies the number of bytes to be written. |

**Figure 88: PCIe Configuration Write – PCIe Request Dword 1**

| Bit | Description |
|---|---|
| 31:12 | Reserved |
| 11:00 | **Offset (OFFSET):** This field specifies the offset in bytes into the 4096B configuration space associated with the NVMe Controller at which the write begins. |

When this command is completed successfully, PCI configuration space associated with the NVMe Controller specified by Controller ID is written with the data contained in the Request Data field. The Offset field specifies the starting write offset in PCIe configuration address space and the Length field specifies the number of bytes to be written. The Request Data field is always an integral number of Dwords and is equal to the Length field rounded up to the next Dword. If Length is not an integral number of Dwords, then unused padding bytes are discarded.

If the sum of the Offset and Length fields fall outside of PCI configuration space, then the Management Endpoint responds with an Invalid Parameter error status response. The parameter with the error in this case is always the Offset field.

## 7.3 PCIe I/O Read

The PCIe I/O Read command allows the Management Controller to read the contents of PCIe I/O space associated with an NVMe Controller in the NVM Subsystem. The Controller ID field in the Command Message specifies the Controller ID that is being accessed.

The command uses PCIe Request Dwords 0 and 1. PCIe Request Dword 2 is not used and is reserved. The format of PCIe Request Dword 0 and 1 are shown in Figure 89 and Figure 90 respectively.

**Figure 89: PCIe I/O Read – PCIe Request Dword 0**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:19</td><td>Reserved</td></tr>
<tr><td>18:16</td><td><b>Base Address Register (BAR):</b> This field specifies the PCI Base Address Register (BAR) of the I/O space to be read. BARs are located beginning at 10h in PCI Configuraiton space and the value of this field specifies the starting offset of the associated BAR. For a 64-bit BAR, this field should correspond to the lower 32-bits of the BAR.
<table>
<tr><th>Value</th><th>BAR Offset</th></tr>
<tr><td>0h</td><td>10h</td></tr>
<tr><td>1h</td><td>14h</td></tr>
<tr><td>2h</td><td>18h</td></tr>
<tr><td>3h</td><td>1Ch</td></tr>
<tr><td>4h</td><td>20h</td></tr>
<tr><td>5h</td><td>24h</td></tr>
<tr><td>6h-7h</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>15:00</td><td>Length (LENGTH): This field specifies the number of bytes to be read.</td></tr>
</table>

**Figure 90: PCIe I/O Read – PCIe Request Dword 1**

| Bit | Description |
|---|---|
| 31:00 | **Offset (OFFSET):** This field specifies the offset in bytes into the PCI BAR associated with the NVMe Controller at which the read begins. |

When this command is completed successfully, PCI I/O space associated with the NVMe Controller specified by Controller ID is read and returned in the Response Data field. The Offset field specifies the starting read offset in PCIe I/O address space specified by the Base Address Register field. The Length field specifies the number of bytes to be read. The Response Data field is always an integral number of Dwords and is equal to the Length field rounded up to the next Dword. If Length is not an integral number of Dwords, then zero padding follows read data.

If the Base Address Register field does not correspond to an I/O BAR implemented by the specified NVMe Controller, then the Management Endpoint responds with an Invalid Parameter error status response.

If the sum of the Offset and Length fields fall outside the address range of the BAR specified by the Base Address Register field, then the Management Endpoint responds with an Invalid Parameter error status response. The parameter with the error in this case is always the Offset field.

### 7.4 PCIe I/O Write

The PCIe I/O Write command allows the Management Controller to write the contents of PCIe I/O space associated with an NVMe Controller in the NVM Subsystem. The Controller ID field in the Command Message specifies the Controller ID that is being accessed.

The command uses PCIe Request Dwords 0 and 1. PCIe Request Dword 2 is not used and is reserved. The format of PCIe Request Dword 0 and 1 are shown in Figure 91 and Figure 92 respectively.

**Figure 91: PCIe I/O Write – PCIe Request Dword 0**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:19</td><td>Reserved</td></tr>
<tr><td>18:16</td><td><b>Base Address Register (BAR):</b> This field specifies the PCI Base Address Register (BAR) of the I/O space to be written. BARs are located beginning at 10h in PCI Configuraiton space and the value of this field specifies the starting offset of the associated BAR. For a 64-bit BAR, this field should correspond to the lower 32-bits of the BAR.
<table>
<tr><th>Value</th><th>BAR Offset</th></tr>
<tr><td>0h</td><td>10h</td></tr>
<tr><td>1h</td><td>14h</td></tr>
<tr><td>2h</td><td>18h</td></tr>
<tr><td>3h</td><td>1Ch</td></tr>
<tr><td>4h</td><td>20h</td></tr>
<tr><td>5h</td><td>24h</td></tr>
<tr><td>6h-7h</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>15:00</td><td>Length (LENGTH): This field specifies the number of bytes to be read.</td></tr>
</table>

**Figure 92: PCIe I/O Write – PCIe Request Dword 1**

| Bit | Description |
|---|---|
| 31:00 | **Offset (OFFSET):** This field specifies the the offset in bytes into the PCI BAR associated with the NVMe Controller at which the write begins. |

When this command is completed successfully, PCI I/O space associated with the NVMe Controller specified by Controller ID is written with the data contained in the Request Data field. The Offset field specifies the starting write offset in PCIe I/O address space specified by the Base Address Register field. The Length field specifies the number of bytes to be written. The Request Data field is always an integral number of Dwords and is equal to the Length field rounded up to the next Dword. If Length is not an integral number of Dwords, then unused padding bytes are discarded.

If the Base Address Register field does not correspond to an I/O BAR implemented by the specified NVMe Controller, then the Management Endpoint responds with an Invalid Parameter error status response.

If the sum of the Offset and Length fields fall outside the address range of the BAR specified by the Base Address Register field, then the Management Endpoint responds with an Invalid Parameter error status response. The parameter with the error in this case is always the Offset field.

### 7.5 PCIe Memory Read

The PCIe Memory Read command allows the Management Controller to read the contents of PCIe memory associated with an NVMe Controller in the NVM Subsystem. The Controller ID field in the Command Message specifies the Controller ID that is being accessed.

The command uses PCIe Request Dwords 0, 1, and 2. The format of PCIe Request Dword 0, 1, and 2 are shown in Figure 93, Figure 94, and Figure 95 respectively.

**Figure 93: PCIe Memory Read – PCIe Request Dword 0**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:19</td><td>Reserved</td></tr>
<tr><td>18:16</td><td><b>Base Address Register (BAR):</b> This field specifies the PCI Base Address Register (BAR) of the memory space to be read. BARs are located beginning at 10h in PCI Configuraiton space and the value of this field specifies the starting offset of the associated BAR. For a 64-bit BAR, this field should correspond to the lower 32-bits of the BAR.
<table>
<tr><th>Value</th><th>BAR Offset</th></tr>
<tr><td>0h</td><td>10h</td></tr>
<tr><td>1h</td><td>14h</td></tr>
<tr><td>2h</td><td>18h</td></tr>
<tr><td>3h</td><td>1Ch</td></tr>
<tr><td>4h</td><td>20h</td></tr>
<tr><td>5h</td><td>24h</td></tr>
<tr><td>6h-7h</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>15:00</td><td>Length (LENGTH): This field specifies the number of bytes to be read.</td></tr>
</table>

**Figure 94: PCIe Memory Read – PCIe Request Dword 1**

| Bit | Description |
|---|---|
| 31:00 | **Offset (OFFSET):** This field specifies the lower 32-bits (i.e., bits 0 through 31) of the offset in bytes into the PCI BAR associated with the NVMe Controller at which the read begins. |

**Figure 95: PCIe Memory Read – PCIe Request Dword 2**

| Bit | Description |
|---|---|
| 31:00 | **Offset (OFFSET):** This field specifies the upper 32-bits (i.e., bits 32 through 63) of the offset in bytes into the PCI BAR associated with the NVMe Controller at which the read begins. |

When this command is completed successfully, PCI memory space associated with the NVMe Controller specified by Controller ID is read and returned in the Response Data field. The Offset field specifies the starting read offset in PCIe memory address space specified by the Base Address Register field. The Length field specifies the number of bytes to be read. The Response Data field is always an integral number of Dwords and is equal to the Length field rounded up to the next Dword. If Length is not an integral number of Dwords, then zero padding follows read data.

If the Base Address Register field does not correspond to one implemented by the specified NVMe Controller, or the address range specified by the Base Address Range is not a memory region, then the Management Endpoint responds with an Invalid Parameter error status response.

If the sum of the Offset and Length fields fall outside the address range specified by the Base Address Register field, then the Management Endpoint responds with an Invalid Parameter error status response. The parameter with the error in this case is always the Offset field.

### 7.6 PCIe Memory Write

The PCIe Memory Write command allows the Management Controller to write the contents of PCIe memory associated with an NVMe Controller in the NVM Subsystem. The Controller ID field in the Command Message specifies the Controller ID that is being accessed.

The command uses PCIe Request Dwords 0, 1, and 2. The format of PCIe Request Dword 0, 1, and 2 are shown in Figure 96, Figure 97, and Figure 98 respectively.

**Figure 96: PCIe Memory Write – PCIe Request Dword 0**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:19</td><td>Reserved</td></tr>
<tr><td>18:16</td><td><b>Base Address Register (BAR):</b> This field specifies the PCI Base Address Register (BAR) of the memory space to be written. BARs are located beginning at 10h in PCI Configuraiton space and the value of this field specifies the starting offset of the associated BAR. For a 64-bit BAR, this field should correspond to the lower 32-bits of the BAR.
<table>
<tr><th>Value</th><th>BAR Offset</th></tr>
<tr><td>0h</td><td>10h</td></tr>
<tr><td>1h</td><td>14h</td></tr>
<tr><td>2h</td><td>18h</td></tr>
<tr><td>3h</td><td>1Ch</td></tr>
<tr><td>4h</td><td>20h</td></tr>
<tr><td>5h</td><td>24h</td></tr>
<tr><td>6h-7h</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>15:00</td><td>Length (LENGTH): This field specifies the number of bytes to be read.</td></tr>
</table>

**Figure 97: PCIe Memory Write – PCIe Request Dword 1**

| Bit | Description |
|---|---|
| 31:00 | **Offset (OFFSET):** This field specifies the lower 32-bits (i.e., bits 0 through 31) of the offset in bytes into the PCI BAR associated with the NVMe Controller at which the write begins. |

**Figure 98: PCIe Memory Write – PCIe Request Dword 2**

| Bit | Description |
|---|---|
| 31:00 | **Offset (OFFSET):** This field specifies the upper 32-bits (i.e., bits 32 through 63) of the offset in bytes into the PCI BAR associated with the NVMe Controller at which the write begins. |

When this command is completed successfully, PCI memory space associated with the NVMe Controller specified by Controller ID is written with the data contained in the Request Data field. The Offset field specifies the starting write offset in PCIe memory address space specified by the Base Address Register field. The Length field specifies the number of bytes to be written. The Request Data field is always an integral number of Dwords and is equal to the Length field rounded up to the next Dword. If Length is not an integral number of Dwords, then unused padding bytes are discarded.

If the Base Address Register field does not correspond to one implemented by the specified NVMe Controller, or the address range specified by the Base Address Range is not a memory region, then the Management Endpoint responds with an Invalid Parameter error status response.

If the sum of the Offset and Length fields fall outside the address range of the BAR specified by the Base Address Register field, then the Management Endpoint responds with an Invalid Parameter error status response. The parameter with the error in this case is always the Offset field.

# 8 NVM Express Management Enhancements

This section describes NVMe Management Interface enhancements to the NVM Express specification.

## 8.1 Identify Controller

The NVMe Identify Controller data structure contains information about an NVMe Controller. Bytes 240-255 have been allocated by the NVM Express specification for NVMe-MI are defined below.

**Figure 99: NVMe Management Interface Identify Controller**

| Bytes | O/M | Description |
|---|---|---|
| 254:240 | | Reserved |
| 255 | M | **Management Endpoint Capabilities (MEC)**: This field indicates the capabilities of the Management Endpoint in the Controller.<br>Bits 7:2 are reserved.<br>Bit 1: If set to '1' then the NVM Subsystem contains a Management Endpoint on a PCIe port.<br>Bit 0: If set to '1' then the NVM Subsystem contains a Management Endpoint on an SMBus/I2C port. |

## 8.2 Management Interface Specific Features

The NVMe Get Features and Set Features Admin commands are used to retrieve and modify Feature values. Feature Identifiers 78h through 7Fh have been allocated by the NVM Express specification for NVMe-MI and are defined by this specification.

**Figure 100: NVMe Management Interface Feature Identifiers**

| Feature Identifier | O/M[1] | Persistent Across Power States and Reset[2] | Uses Memory Buffer for Attributes | Description |
|---|---|---|---|---|
| 78h – 7Dh | | | | Reserved |
| 7Eh | M | No | Yes | Controller Metadata |
| 7Fh | M | No | Yes | Namespace Metadata |

NOTES:
1. O/M definition: O = Optional, M = Mandatory. Mandatory commands shall be supportd if the NVM Subsystem implements a Management Endpoint. These features are not mandatory if the subsystem does not implement a Management Endpoint.
2. This column is only valid if bit 4 in the Optional NVM Command Support field of the Identify Controller Data structure is cleared to '0'. Refer to the NVMe specification.

### 8.2.1 Controller Metadata

This feature is used to store metadata about the host platform in an NVM Subsystem for later retrieval. The values stored in the Controller Metadata Feature do not modify Controller behavior.

The Controller Metadata feature uses NVMe Set Feature Command Dword 11 as shown in Figure 101.

**Figure 101: Host Metadata – Command Dword 11**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:15</td><td>Reserved</td></tr>
<tr><td>14:13</td><td><p><b>Element Action (EA):</b> This field specifies the action to perform on the Metadata Element Descriptor data structure. This field shall be cleared to 0h for a Get Features.</p>
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00b</td><td>Add/Update Entry</td></tr>
<tr><td>01b</td><td>Delete Entry</td></tr>
<tr><td>10 - 11b</td><td>Reserved</td></tr>
</table>
<p>If the Element Action field is set to 00b (Add/Update Entry) and a Metadata Element Descriptor with the specified Element Type does not already exist in Controller Metadata, then the Controller creates a new descriptor with the value in the Controller Metadata structure. This operation is performed in an atomic manner.</p>
<p>If the Element Action field is set to 00b (Add/Update Entry) and a Metadata Element Descriptor with the specified Element Type already exists in the Controller Metadata, then the Controller updates the descriptor with the value in the Controller Metadata structure. This operation is performed in an atomic manner.</p>
<p>If the Element Action field is set to 01b (Delete Entry) and a Metadata Element Descriptor with the specified Element Type does not already exists in the Controller Metadata, then no operation is performed and the command completes successfully.</p>
<p>If the Element Action field is set to 01b (Delete Entry) and a Metadata Element Descriptor with the specified Element Type already exists in the Controller Metadata, then the Controller deletes the specified Metadata Element Descriptor. This operation is performed in an atomic manner.</p></td></tr>
<tr><td>12:00</td><td>Reserved</td></tr>
</table>

New metadata elements may be added, updated, or deleted based on the action specified in the Element Action field.

If a Set Features command is submitted for this Feature, a Host Metadata data structure, defined in Figure 102, is transferred in the data buffer for the command. The Host Metadata data structure is 4096 bytes in size and contains one or more Metadata Element Descriptors. If host software attempts to add or update a Metadata Element that causes the stored Host Metadata data structure to grow larger than 4096 bytes, the Controller shall abort the command with the status code Invalid Field in Command. The Host Metadata structure for this feature is independent of the Host Metadata data structure for the Namespace Metadata feature described in section 8.2.2.

**Figure 102: Host Metadata Data Structure**

| Byte | Description |
|---|---|
| 0 | **Number of Metadata Element Descriptors:** This field contains the number of Metadata Elements in the data structure. |
| 1 | Reserved |
| x:2 | **Metadata Element Descriptor 0:** This field contains the first Metadata Element descriptor. |
| y:x+1 | **Metadata Element Descriptor 1:** This field contains the second Metadata Element descriptor or 0h if there is only 1 entry. |
| … | … |
| 4095:z | **Metadata Element Descriptor N:** This field contains the N+1 th Metadata Element descriptor or 0h if there are fewer than N+1 entries. |

A Host Metadata data structure may contain at most one Metadata Element Descriptor of each element type. Each Metadata Element Descriptor contains the data structure shown in Figure 103.

**Figure 103: Metadata Element Descriptor**

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>32 + (Element Length*8) :32</td><td><b>Element Value (EVAL):</b> This field specifies the value for the element.</td></tr>
<tr><td>31:16</td><td><b>Element Length (ELEN):</b> This field specifies the length of the Element Value field in bytes. This field shall be 0h when deleting an entry (EA = 01b).</td></tr>
<tr><td>15:12</td><td>Reserved</td></tr>
<tr><td>11:8</td><td><b>Element Revision (ER):</b> This field specifies the revision of this element value. Unless specified otherwise elsewhere in this specification, all Metadata Element Descriptors compliant with this version of the NVMe-MI Specifciation shall set this field to a value of 0h.</td></tr>
<tr><td>7:6</td><td>Reserved</td></tr>
<tr><td>5:0</td><td><b>Element Type (ET):</b> This field specifies the type of metadata stored in the descriptor.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00h</td><td>Reserved</td></tr>
<tr><td>01h – 017h</td><td>NVMe-MI defined element types. Controller Metadata Element types are defined in Figure 104. Namespace Metadata Element types are defined in Figure 105.</td></tr>
<tr><td>18h – 1Fh</td><td>Vendor Specific</td></tr>
</table>
</td></tr>
</table>

If a Get Features command is issued for this Feature, all Controller Metadata associated with the specified Controller is added to a Host Metadata Data Structure specified in Figure 102 and returned in the data buffer for that command. The data buffer size is equal to the size of the Host Metadata Data Structure and is 4096 bytes in size.

**Figure 104: Controller Metadata Element Types**

| Value | Definition |
|---|---|
| 00h | Reserved |
| 01h | **Operating System Controller Name:** The name of the Controller in the operating system as a UTF-8 string. |
| 02h | **Operating System Driver Name:** The name of the driver in the operating system as a UTF-8 string. |
| 03h | **Operating System Driver Version:** The version of the driver in the operating system as a UTF-8 string. |
| 04h | **Pre-boot Controller Name:** The name of the driver in the pre-boot environment as a UTF-8 string. |
| 05h | **Pre-boot Driver Name:** The name of the driver in the pre-boot environment as a UTF-8 string. |
| 06h | **Pre-boot Driver Version:** The version of the driver in the pre-boot environment as a UTF-8 string. |
| 07h – 17h | Reserved |
| 18h – 1Fh | Vendor Specific |

Controller Metadata is reset on a Controller Level Reset (i.e., the number of stored Metadata Element Descriptors is zero). Executing a Get Features command while the Controller is disabled returns zero Metadata Element Descriptors.

### 8.2.2 Namespace Metadata

This feature is used to store metadata about a namespace associated with a Controller in the NVM Subsystem for later retrieval. The values stored in the Namespace Metadata Feature do not modify Controller behavior on the namespace. This feature is namespace specific.

The Namespace Metadata feature uses Command Dword 11 as shown in Figure 101.

New metadata elements may be added, updated, or deleted based on the action specified in the Element Action field.

If a Set Features command is submitted for this Feature, a Host Metadata data structure, defined in Figure 102, is transferred in the data buffer for the command. The Host Metadata data structure is 4096 bytes in size and contains one or more Metadata Element Descriptors. If host software attempts to add or update a Metadata Element that causes the stored Host Metadata data structure to grow larger than 4096 bytes, the Controller shall abort the command with the status code Invalid Field in Command. The Host Metadata structure for this feature is independent of the Host Metadata data structure for the Controller Metadata feature described in section 8.2.1.

A Host Metadata data structure may contain up to one Metadata Element Descriptor of each element type. Each Metadata Element Descriptor contains the data structure shown in Figure 103.

If a Get Features command is issued for this Feature, all Namespace Metadata associated with the specified Controller is added to a Host Metadata Data Structure specified in Figure 102 and returned in the data buffer for that command. The data buffer size is equal to the size of the Host Metadata Data Structure and is 4096 bytes in size.

Namespace Metadata is reset on a Controller Level Reset (i.e., the number of stored Metadata Element Descriptors is zero). Executing a Get Features command while the Controller is disabled returns zero Metadata Element Descriptors.

**Figure 105: Namespace Metadata Element Types**

| Value | Definition |
|---|---|
| 00h | Reserved |
| 01h | **Operating System Namespace Name:** The name of the namespace in the operating system as a UTF-8 string. |
| 02h | **Pre-boot Namespace Name:** The name of the namespace in the pre-boot environment as a UTF-8 string. |
| 03h – 17h | Reserved |
| 18h – 1Fh | Vendor Specific |

# 9 Management Architecture

## 9.1 Operational Times

The ability of a Management Endpoint to receive and process Request Messages outlined in this specification is dependent on the state of the Management Endpoint. This section enumerates Management Endpoint operational times and the operations supported in each of these operational times.

The NVM Subsystem power state is defined by the state of main power and auxiliary power. Main power consist of one or more voltage rails as defined by form factor. When main power consists of multiple voltage rails, main power is considered "on" when power is good on all main voltage rails. Auxiliary power is optionally supported by a form factor and enables wake-up processing in the absence of main power. Auxiliary power is considered "off" in form factors and platforms that do not support auxiliary power. Figure 106 defines the power states of a Management Endpoint.

**Figure 106: NVM Subsystem Power States**

| Power State | Main Power | Auxiliary Power |
|---|---|---|
| Powered Off | Off | Off |
| Auxiliary Power | Off | On |
| Main Power | On | On |
| Main Power with No Auxiliary Power | On | Off |

The operations supported in each NVM Subsystem power state are summarized in Figure 107. VPD SMBus/I2C access consists of processing read operations to the FRU Information Device. SMBus/I2C MCTP access consists of processing and responding to MCTP messages and responding to the NVMe Basic Management Command (refer to Appendix A) on the NVM Subsystem SMBus/I2C port. PCIe MCTP access consists of processing and responding to MCTP messages issued on any NVM Subsystem PCIe port. The behavior of an operation that is "Not Supported" in Figure 107 is undefined.

**Figure 107: Operations Supported During NVM Subsystem Power States**

| Operation | Powered Off | Auxiliary Power | Main Power (with Auxiliary Power) | Main Power with No Auxiliary Power |
|---|---|---|---|---|
| VPD I2C Access | Not Supported | Supported | Supported | Inplementation Specific |
| SMBus/I2C MCTP Access | Not Supported | Optional[1] | Supported | Supported |
| PCIe MCTP Access | Not Supported | Not Supported | Supported | Supported |

When an NVM Subsystem transitions from a power state in which accesses are not supported to one where accesses are supported, accesses shall be processed one second after entering the power state in which accesses are supported. For example, an SMBus/I2C MCTP access issued one second after transitioning from a “Powered Off” to a “Main Power” state is guaranteed to be processed. The behavior of accesses prior to this one second time interval is undefined. For example, the behavior of an SMBus/I2C MCTP access issued 50ms after transitioning from a “Powered Off” to a “Main Power” state is undefined.

When transitioning between power states in which accesses are supported in both states (i.e., the state before and after the transition), there is no interruption in access processing (i.e., accesses are processed prior to the state transition, during the state transition, and immediately after entering the new power state).

Request Messages are processed whenever MCTP access is supported on an interface (i.e., SMBus/I2C or PCIe). Although not recommended, an implementation may not support PCIe and SMBus/I2C MCTP accesses during a PCI Express conventional reset on any PCI Express port in the NVM Subsystem. Although not recommended, an implementation may choose not to support processing of PCIe Commands that target a Controller in the NVM Subsystem that is in one of the following states:[2]

- Controller Level Reset
- SR-IOV virtual function is not enabled,
- During any type of PCI Express Conventional Reset,
- During a PCI Express Function Level Reset (FLR),
- When the PCI Express Function is in a non-D0 power D-state, and
- When the PCI Express link is down (i.e., not in the DL_Active state).

If a PCIe Command is received that targets a Controller in one of these states and the implementation does

[1] An implementation that supports SMBus MCTP Access during Auxiliary Power may support a subset of commands during this power state. The commands that are supported are implementation specific.
[2] A Management Controller shall only send these commands using SMBus/I2C or another PCIe port since the link associated with the PCIe port and controller is down in these states.

not support processing of PCIe Commands in that state, then the PCIe command is completed with status PCIe Inaccessible. Processing of supported PCIe Commands is required in all other Controller states.

If a PCIe Command is received that targets a Controller whose corresponding PCIe link is in a low power state (i.e., PCIe ASPM), then processing of the command may cause the link to temporarily exit the low power state.

## 9.2 Vital Product Data

Each NVM Subsystem with one or more Management Endpoints shall have a FRU Information Device which is compliant with the IPMI Platform Management FRU Information Storage Definition. The VPD shall contain the required elements defined in Figure 108. The size of the VPD is 256 bytes as defined by the IPMI Platform Management FRU Information Storage Definition.

The VPD shall be accessible using the VPD Read command. The entire contents of the VPD may be updated using the VPD Write command.

If the NVM Subsystem has an SMBus/I2C interface, the VPD shall be accessible at the SMBus/I2C address of the FRU Information Device using the access mechanism over I2C as defined in the IPMI Platform Management FRU Information Storage Definition. Updating the VPD by writing to the FRU Information Device directly on SMBus/I2C shall not be supported.

**Figure 108: VPD Elements**

| Byte | Name |
|---|---|
| 07:00 | Common Header |
| 119:08 | Product Info Area |
| Vendor Specific:120 | MultiRecord Info Area |
| Vendor Specific | Internal Use Area (optional) |
| Vendor Specific | Chassis Info Area (optional) |
| Vendor Specific | Board Info Area (optional) |

VPD records utilize the Type/Length byte format defined in the IPMI Platform Management FRU Information Storage Definition. Type/Length byte encodings utilized in this specification are summarized in Figure 109.

**Figure 109: Type/Length Byte Format**

| Bits | Field Name | Description |
|---|---|---|
| 7:6 | Type Code | Specifies field encoding<br>11b – Always corresponds to ASCII in this specification |
| 5:0 | Number of Data Bytes | Specifies field length<br>000000b indicates that the field is empty |

### 9.2.1 Common Header

| Byte | Factory Default | Description |
|---|---|---|
| 00 | 01h | **IPMI Format Version Number (IPMIVER)**: This field indicates the IPMI Format Version. |
| 01 | Impl Spec | **Internal Use Area Starting Offset (IUAOFF):** This field indicates the starting offset in multiples of 8 bytes for the Internal Use Area. A value of 00h may be used to indicate the Internal Use Area is not present. |
| 02 | Impl Spec | **Chassis Info Area Starting Offset (CIAOFF)**: This field indicates the starting offset in multiples of 8 bytes for the Chassis Info Area. A value of 00h may be used to indicate the Chassis Info Area is not present. |

| | | |
|---|---|---|
| 03 | Impl Spec | **Board Info Area Starting Offset (BIAOFF)**: This field indicates the starting offset in multiples of 8 bytes for the Board Info Area. A value of 00h may be used to indicate the Board Info Area is not present. |
| 04 | 01h | **Product Info Area Starting Offset (PIAOFF)**: This field indicates the starting offset in multiples of 8 bytes for the Product Info Area. |
| 05 | 0Fh | **MultiRecord Info Area Starting Offset (MRIOFF):** This field indicates the starting offset in multiples of 8 bytes for the MultiRecord Info Area. |
| 06 | 00h | Reserved |
| 07 | Impl Spec | **Common Header Checksum (CHCHK):** Checksum computed over bytes 0 through 6. The checksum is computed by adding the 8-bit value of the bytes modulo 256 and then taking the 2’s complement of this sum. When the checksum and the sum of the bytes module 256 are added, the result should be 0h. |

### 9.2.2 Product Info Area (offset 8 bytes)

| Byte Offset | Factory Default | Description |
|---|---|---|
| 00 | 01h | **IPMI Format Version Number (IPMIVER)**: This field indicates the IPMI Format Version. |
| 01 | 0Eh | **Product Info Area Length (PALEN)**: This field indicates the length of the product info area in multiples of 8 bytes. 112 bytes/8 = 14 = 0x0Eh |
| 02 | 19h | **Language Code (LCODE):** This field indicates the language used. A value of 19h is used to indicate English. |
| 03 | C8h | **Manufacturer Name Type/Length (MNTL):** This byte indicates the type and length of the Manufacturer Name field. |
| 11:04 | Impl Spec | **Manufacturer Name (MNAME):** This field indicates the Manufacturer name in 8-bit ASCII. Unused bytes should be NULL characters.<br>The Manufacturer name in this field should correspond to that in the PCI Subsystem Vendor ID (SSVID) and IEEE OUI Identifier fields in the Identify Controller Data Structure |
| 12 | D8h | **Product Name Type/Length (PNTL):** This byte indicates the type and length of the Product Name field. |
| 36:13 | Impl Spec | **Product Name (PNAME):** This field indicates the Product name in 8-bit ASCII. Unused bytes should be NULL characters. |
| 37 | E8h | **Product Part/Model Number Type/Length (PPMNNTL):** This byte indicates the type and length of the Product Part/Model Number field. |
| 77:38 | Impl Spec | **Product Part/Model Number (PPMN):** This field indicates the Product Part/Model Number in 8-bit ASCII. Unused bytes should be NULL characters.<br>This field should contain the same value as the Model Number (NM) field in the NVMe Identify Controller Data Structure |
| 78 | C2h | **Product Version Type/Length (PVTL):** This byte indicates the type and length of the Product Part/Model Number field. |
| 80:79 | Impl Spec | **Product Version (PVER):** This field indicates the Product Version in 8-bit ASCII. Unused bytes should be NULL characters. |
| 81 | D4h | **Product Serial Number Type/Length (PSNTL):** This byte indicates the type and length of the Product Serial Number field. |
| 101:82 | Impl Spec | **Product Serial Number (PSN):** This field indicates the Product Serial Number in 8-bit ASCII. Unused bytes should be NULL characters.<br>This field should contain the same value as the Serial Number (SN) field in the NVMe Identify Controller Data Structure. |
| 102 | 0h | **Asset Tag Type/Length (ATTL):** This byte indicates the type and length of the Asset Tag field. A value of 00h may be used to indicate an Asset Tag is not present. |
| 103 | 0h | **FRU File ID Type/Length (ATTL):** This byte indicates the type and length of the FRU File ID field. A value of 00h may be used to indicate a FRU File ID is not present. |

| | | |
|---|---|---|
| 104 | C1h | **End of Record (EOR)**: A value of C1h in this byte indicates the end of record |
| 110:105 | | Reserved |
| 111 | Impl Spec | **Product Info Area (PICHK):** Checksum computed over bytes 0 through 110. The checksum is computed by adding the 8-bit value of the byes modulo 256 and then taking the 2's complement of this sum. When the checksum and the sum of the bytes module 256 are added, the result should be 0h. |

### 9.2.3 NVMe MultiRecord Area

| Byte Offset | Factory Default | Description |
|---|---|---|
| 00 | 0Bh | **NVMe Record Type ID** |
| 01 | 2h | Bit 7 – end of list; record format version = 2h |
| 02 | 28h | **Record Length (RLEN)**: This field indicates the length of the MultiRecord Area in bytes. |
| 03 | Impl Spec | **Record Checksum**: This field is used to give the record data a zero checksum (i.e., the modulo 256 sum of the record data bytes from byte offset 05 through the end of this record plus this checksum byte equals zero) |
| 04 | Impl Spec | **Header Checksum**: This field is used to give the record header a zero checksum (i.e., the modulo 256 sum of the preceding record bytes starting with the first byte of the header plus this checksum byte equals zero. |
| 05 | 0h | **NVMe MultiRecord Area Version Number:** This field indicates the version number of this multirecord. This field shall be set to 0h in this version of the specification. |
| 06 | Impl Spec | **Management Endpoint Form Factor (MEFF):** This field indicates the form factor of the Management Endpoint. |

| Value | Definition |
|---|---|
| 0 | Other – unknown |
| 1 – 15 | Reserved |
| 16 | 2.5" Form Factor – unknown |
| 17 | 2.5" Form Factor – U.2 (SFF-8639) 15mm |
| 18 | 2.5" Form Factor – U.2 (SFF-8639) 7mm |
| 19 – 31 | Reserved |
| 32 | CEM add in card – unknown |
| 33 | CEM add in card – Low Profile (HHHL) |
| 34 | CEM add in card – Standard Height Half Length (FHHL) |
| 35 | CEM add in card – Standard Height Full Length (FHFL) |
| 36-47 | Reserved |
| 48 | M.2 module – unknown |
| 49 | M.2 module – 2230 |
| 50 | M.2 module – 2242 |
| 51 | M.2 module – 2260 |
| 52 | M.2 module – 2280 |
| 53 | M.2 module – 22110 |
| 54-63 | Reserved |
| 64 | BGA SSD – unknown |
| 65-239 | Reserved |
| 240-255 | Vendor Specific |

| | | |
|---|---|---|
| 12:07 | | Reserved |
| 13 | Impl Spec | **Initial 1.8V Power Supply Requirements:** This field specifies the initial 1.8V power supply requirements in Watts prior to receiving a Set Slot Power message. |
| 14 | Impl Spec | **Maximum 1.8V Power Supply Requirements:** This field specifies the maximum 1.8V power supply requirements in Watts. A value of zero indicates that the power supply voltage is not used. |
| 15 | Impl Spec | **Initial 3.3V Power Supply Requirements:** This field specifies the initial 3.3V power supply requirements in Watts prior to receiving a Set Slot Power message. |
| 16 | Impl Spec | **Maximum 3.3V Power Supply Requirements:** This field specifies the maximum 3.3V power supply requirements in Watts. A value of zero indicates that the power supply voltage is not used. |
| 17 | | Reserved |
| 18 | Impl Spec | **Maximum 3.3V aux Power Supply Requirements:** This field specifies the maximum 3.3V power supply requirements in 10 mW units. A value of zero indicates that the power supply voltage is not used. |
| 19 | Impl Spec | **Initial 5V Power Supply Requirements:** This field specifies the initial 5V power supply requirements in Watts prior to receiving a Set Slot Power message. |
| 20 | Impl Spec | **Maximum 5V Power Supply Requirements:** This field specifies the maximum 5V power supply requirements in Watts. A value of zero indicates that the power supply voltage is not used. |
| 21 | Impl Spec | **Initial 12V Power Supply Requirements:** This field specifies the initial 12V power supply requirements in Watts prior to receiving a Set Slot Power message. |
| 22 | Impl Spec | **Maximum 12V Power Supply Requirements:** This field specifies the maximum 12V power supply requirements in Watts. A value of zero indicates that the power supply voltage is not used. |
| 23 | Impl Spec | **Maximum Thermal Load:** This field specifies the maximum thermal load from the NVM Subsystem in Watts. |
| 36:24 | Impl Spec | **Total NVM Capacity:** This field indicates the total NVM capcity of the Management Endpoint in bytes.<br>If the NVM Subsystem supports Namespace Management, then this field should correspond to the value reported in the TNVMCAP field in the NVMe Identify Controller Data structure.<br>A value of 0h may be used to indicate this feature is not supported. |
| 63:37 | | Reserved |

### 9.2.4 NVMe PCIe Port MultiRecord Area

| Byte Offset | Factory Default | Description |
|---|---|---|
| 00 | 0Ch | **NVMe PCIe Port Record Type ID** |
| 01 | 2h | Bit 7 – end of list; record format version = 2h |
| 02 | 28h | **Record Length (RLEN)**: This field indicates the length of the MultiRecord Area in bytes. |
| 03 | Impl Spec | **Record Checksum**: This field is used to give the record data a zero checksum (i.e., the modulo 256 sum of the record data bytes from byte offset 05 through the end of this record plus this checksum byte equals zero) |

<table>
<tr><td>04</td><td>Impl Spec</td><td><b>Header Checksum</b>: This field is used to give the record header a zero checksum (i.e., the modulo 256 sum of the preceding record bytes starting with the first byte of the header plus this checksum byte equals zero.</td></tr>
<tr><td>05</td><td>0h</td><td><b>NVMe PCIe Port MultiRecord Area Version Number:</b> This field indicates the version number of this multirecord. This field shall be set to zero in this version of the specification.</td></tr>
<tr><td>06</td><td>Impl Spec</td><td><b>PCIe Port Number:</b> This field contains the PCIe port number. This is the same value as that reported in the Port Number field in the PCIe Link Capabilities Register.</td></tr>
<tr><td>07</td><td>Impl</td><td><b>Port Information:</b> This field indicates information about the PCIe Ports in the device.<br>Bits 7 to 1 are reserved.<br>Bit 0, if set to ‘1’ indicates that all PCIe ports within the device have the same capabilities (i.e., the capabilities listed in this structure are consistent across each PCIe port).</td></tr>
<tr><td>08</td><td>Impl Spec</td><td><b>PCIe Link Speed:</b> This field indicates a bit vector of link speeds supported by the PCIe port.
<table>
<tr><th>Bit</th><th>Definition</th></tr>
<tr><td>7:3</td><td>Reserved</td></tr>
<tr><td>2</td><td>Set to ‘1’ if the PCIe link supports 8.0 GT/s. Otherwise cleared to ‘0’.</td></tr>
<tr><td>1</td><td>Set to ‘1’ if the PCIe link supports 5.0 GT/s. Otherwise cleared to ‘0’.</td></tr>
<tr><td>0</td><td>Set to ‘1’ if the PCIe link supports 2.5 GT/s. Otherwise cleared to ‘0’.</td></tr>
</table></td></tr>
<tr><td>09</td><td>Impl Spec</td><td><b>PCIe Maximum Link Width:</b> The maximum PCIe link width for this NVM Subsystem port. This is the expected negotiated link width that the port link trains to if the platform supports it. A Management Controller may compare this value with the PCIe Negotiated Link Width to determine if there has been a PCIe link training issue.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>PCIe x1</td></tr>
<tr><td>2</td><td>PCIe x2</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
<tr><td>4</td><td>PCIe x4</td></tr>
<tr><td>5-7</td><td>Reserved</td></tr>
<tr><td>8</td><td>PCIe x8</td></tr>
<tr><td>9-11</td><td>Reserved</td></tr>
<tr><td>12</td><td>PCIe x12</td></tr>
<tr><td>13-15</td><td>Reserved</td></tr>
<tr><td>16</td><td>PCIe x16</td></tr>
<tr><td>17-31</td><td>Reserved</td></tr>
<tr><td>32</td><td>PCIe x32</td></tr>
<tr><td>33-255</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>10</td><td>Impl Spec</td><td><b>MCTP Support:</b> This field contains a bit vector that specifies the level of support for the NVMe Management Interface.<br>Bits 7 to 1 are reserved.</td></tr>
</table>

<table>
<tr><td></td><td></td><td>Bit 0, if set to ‘1’ indicates that MCTP based management commands are supported on the PCIe port.</td></tr>
<tr><td>11</td><td>Impl Spec</td><td><b>Ref Clk Capability:</b> This field contains a bit vector that specifies the PCIe clocking modes supported by the port.
<table>
<tr><th>Bit</th><th>Definition</th></tr>
<tr><td>7:4</td><td>Reserved</td></tr>
<tr><td>3</td><td>Set to ‘1’ if the device automatically uses RefClk if provided and otherwise uses SRIS. Otherwise cleared to ‘0’.</td></tr>
<tr><td>2</td><td>Set to ‘1’ if the PCIe link supports Separate ReClk with SSC (SRIS). Otherwise cleared to ‘0’.</td></tr>
<tr><td>1</td><td>Set to ‘1’ if the PCIe link supports Separate ReClk with no SSC (SRNS). Otherwise cleared to ‘0’.</td></tr>
<tr><td>0</td><td>Set to ‘1’ if the PCIe link supports common ReClk. Otherwise cleared to ‘0’.</td></tr>
</table></td></tr>
<tr><td>15:12</td><td>00h</td><td>Reserved</td></tr>
</table>

## 9.3 Reset

This section describes NVMe-MI architected resets.

### 9.3.1 NVM Subsystem Reset

An NVM Subsystem Reset is initiated under the conditions outlined in the NVMe specification (e.g., when main power is applied to the NVM Subsystem). In addition to these conditions, an NVM Subsystem Reset may be initiated by executing a Reset command.

An NVMe-MI initiated NVM Subsystem Reset may interfere with host software. A Management Controller should coordinate with the host. Coordination between a Management Controller and a host are outside the scope of this specification.

When an NVM Subsystem Reset is initiated, the entire NVM Subsystem is reset. This includes all NVM Subsystem port (PCIe and SMBus/I2C), Management Endpoints, and Controller Management Interfaces. All state is returned to its default condition.

### 9.3.2 Controller Level Reset

A Controller Level Reset is initiated under the conditions outlined in the NVMe specification.

An NVMe-MI initiated Controller Level Reset may interfere with host software. A Management Controller should coordinate with the host. Coordination between a Management Controller and a host are outside the scope of this specification.

The actions performed on a Controller Level Reset are outlined in the NVMe specification. A Controller Level Reset has no effect on the Controller Management Interface associated with that Controller, the PCI Express port associated with that Controller, or a Management Endpoint associated with that port. A Controller Level Reset also has no effect on Management Interface Command Set or NVM Express Admin Command Set commands that target that Controller (i.e., the NVM Express Admin Command Set is still available even though the NVMe Controller may be disabled or held in reset) or Control Primitives. A Controller Level Reset may affect PCIe Command Set commands executing on that Controller (refer to 9.1). If a PCIe Command is affected, then the command is completed with status PCIe Inaccessible.

A Controller Level Reset that causes a new firmware image to activate is considered a special event and may impact the operation of the Controller Management Interface associated with one or more Controllers, execution of NVMe-MI commands, and Management Endpoints within an NVM Subsystem. This impact is unspecified and vendor specific. The Management Controller and host should coordinate the activation of a new firmware image. Coordination between a Management Controller and a host are outside the scope of this specification.

### 9.3.3 Management Endpoint Reset

A Management Endpoint reset is initiated under the conditions outlined in the MCTP Base Specification or the associated MCTP transport binding specifications.

In addition to these conditions, a Management Endpoint associated with a PCI Express port is reset when the PCI Express port is in one of the following states:

- A PCI Express conventional reset, and
- When the PCI Express link is down (i.e., not in the DL_Active state).

When a Management Endpoint Reset is initiated, the state of that Management Endpoint is returned to its default condition and any commands associated with that Management Endpoint being processed are aborted. A reset of a Management Endpoint in an NVM Subsystem has no effect on any other Management Endpoint in the NVM Subsystem or any other NVM Subsystem entity.

## 9.4 Security

The Management Endpoint may respond with a Response Message Status value of Access Denied in an error response. While a drive is in an unlocked state, this mechanism may not be used for the Management Interface Command Set or the NVMe Admin Command Set.

The commands and the times at which such a response is generated is vendor specific. The mechanism used to lock a drive is outside the scope of this specification.

# Appendix A – Technical Note: NVM Express Basic Management Command

This specification utilizes Management Component Transport Protocol (MCTP) messages. The NVMe Basic Management Command does not use MCTP. Support for the NVMe Basic Management Command is optional.

This command does not provide any mechanism to modify or configure the NVMe device. Such features use the more capable MCTP protocol rather than this command's simpler SMBus Block Read. The host may reuse existing SMBus or FRU Information Device read subroutines for this read and is not required to switch the SMBus between master and slave modes as in MCTP.

The block read protocol is specified by the SMBus specification which is available online at www.smbus.org. First slave address write and command code bytes are transmitted by the host, then a repeated start and finally a slave address read. The host keeps clocking as the drive then responds in slave mode with the selected data. The command code is used as a starting offset into the data block shown in Figure 110, like an address on a serial EEPROM.

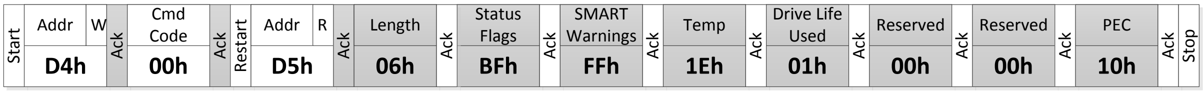
The offset value increments on every byte read and is reset to zero on a stop condition. A read command without a repeated start is permissible and starts transmission from offset zero. Reading more than the block length with an I2C read is also permissible and these reads continue into the first byte in the next block of data. The PEC accumulates all bytes sent or received after the start condition and the current value is inserted whenever a PEC field is reached.

Blocks of data are packed sequentially. The first 2 blocks are defined by the NVMe-MI workgroup. The first block is the dynamic host health data. The second block includes the Vendor ID (VID) and serial number of the drive. Additional blocks of data may be defined by the owner of the VID. Reading past the end of the vendor defined blocks shall return zeros.

The SMBus slave address to read this data structure is not the same address we use for MCTP, and defaults to 6Ah if ARP is not invoked[1]. Since SMBus shifts the address left to make room for the read/write direction bit, the address appears in the examples below as D4h for write and D5h for read. Interleaved MCTP and block read traffic is permissible and neither command type shall disturb the state of the other commands.

Here are a few example reads from an NVMe drive at 30°C, no alarms, VID=1234h, serial number is AZ123456 using the format defined in Figure 110. Host transmissions are shown in white blocks and drive responses are shown in grey blocks:

**Example 1:** SMBus block read of the drive's status (status flags, SMART warnings, temperature):



| Start | Addr | W | Ack | Cmd Code | Ack | Restart | Addr | R | Ack | Length | Ack | Status Flags | Ack | SMART Warnings | Ack | Temp | Ack | Drive Life Used | Ack | Reserved | Ack | Reserved | Ack | PEC | Ack | Stop |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **D4h** | | | **00h** | | | **D5h** | | | **06h** | | **BFh** | | **FFh** | | **1Eh** | | **01h** | | **00h** | | **00h** | | **10h** | | |

**Example 2:** SMBus block read of the drive's static data (VID and serial number):

[1] Note that a previous version of this command mentioned that it would be the same address as MCTP.

| Start | Addr | W | Ack | Cmd Code | Ack | Restart | Addr | R | Ack | Length | Ack | VID | Ack | VID | Ack | Serial # ‘A’ | Ack | Serial # ‘Z’ | Ack | Serial # ‘1’ | Ack | Serial # ‘2’ | Ack | Serial # ‘3’ | Ack | Serial # ‘4’ | Ack |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | D4h | | | 08h | | | D5h | | | 16h | | 12h | | 34h | | 41h | | 5Ah | | 31h | | 32h | | 33h | | 34h | |

| Serial # ‘5’ | Ack | Serial # ‘6’ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 35h | | 36h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | |

| Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | PEC | Ack | Stop |
|---|---|---|---|---|---|---|
| 20h | | 20h | | DAh | | |

**Example 3:** SMBus send byte to reset Arbitration bit:

| Start | Addr | W | Ack | Cmd Code | Ack | Stop |
|---|---|---|---|---|---|---|
| | D4h | | | FFh | | |

**Example 4:** I2C read of status and vendor content, I2C allows reading across SMBus block boundaries:

| Start | Addr | W | Ack | Cmd Code | Ack | Restart | Addr | R | Ack | Length | Ack | Status Flags | Ack | SMART Warnings | Ack | Temp | Ack | Drive Life Used | Ack | Reserved | Ack | Reserved | Ack | PEC | Ack | Length | Ack |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | D4h | | | 00h | | | D5h | | | 06h | | BFh | | FFh | | 1Eh | | 01h | | 00h | | 00h | | 10h | | 16h | |

| VID | Ack | VID | Ack | Serial # ‘A’ | Ack | Serial # ‘Z’ | Ack | Serial # ‘1’ | Ack | Serial # ‘2’ | Ack | Serial # ‘3’ | Ack | Serial # ‘4’ | Ack | Serial # ‘5’ | Ack | Serial # ‘6’ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12h | | 34h | | 41h | | 5Ah | | 31h | | 32h | | 33h | | 34h | | 35h | | 36h | | 20h | | 20h | |

| Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | Serial # ‘ ‘ | Ack | PEC | Ack | Stop |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | 20h | | B0h | | |

The SMBus Arbitration bit may be used for simple arbitration on systems that have multiple drives on the same SMBus segment without ARP or muxes to separate them. To use this mechanism, the host follows this 3 step process to handle collisions for the same slave address:

1. The host does an SMBus byte write to send byte FFh which clears the SMBus Arbitration bit on all listening Management Endpoints at this slave address.
2. The host does an I2C read starting from offset 0h and continuing at least through the serial number in the second block. The drive transmitting a ‘0’ when other drives sent a ‘1’ wins arbitration and sets the arbitration bit to ‘1’ upon read completion to give other drives priority on the next read.
3. Repeat step 2 until all drives are read, host receiving the Arbitration bit as a ‘1’ indicates loop is done.
4. Sort the responses by serial number since the order of drive responses varies with health status and temperatures.

Be careful that there are no short reads of similar data between steps 1 and 3. If the read data is exactly the same on multiple drives then all these drives set the arbitration bit. After that a new send byte FFh is required to restart the process.

The logic levels were intentionally inverted to normally high in the bytes 1 and 2. This is an additional mechanism to assist systems that do not have ARP or muxes. Since ‘0’ bits win arbitration on SMBus, a drive with an alarm condition is prioritized over healthy drives in the above arbitration scheme. A single I2C read of byte offsets 1&2 from an array of drives detects alarm conditions. Note that only one drive with an alarm may be reliably detected because drives without the same alarm stop transmitting once the bus contention is detected. For this reason the bits are sorted in order of priority. Continuing to read further provides the serial number of the drive that had the alarm.

**Figure 110: Subsystem Management Data Structure**

| Command Code | Offset (byte) | Description |
|---|---|---|
| 0 | 00 | **Length of Status:** Indicates number of additional bytes to read before encountering PEC. This value should always be 6 (06h) in implementations of this version of the spec. |
| | 01 | **Status Flags (SFLGS):** This field indicates the status of the NVM Subsystem.<br>**SMBus Arbitration** – Bit 7 is set ‘1’ after an SMBus block read is completed all the way to the stop bit without bus contention and cleared to ‘0’ if an SMBus Send Byte FFh is received on this SMBus slave address.<br>**Drive Not Ready** – Bit 6 is set to ‘1’ when the subsystem is not capable of processing NVMe management commands, and the rest of the transmission may be invalid. If cleared to ‘0’, then the NVM Subsystem is fully powered and ready to respond to management commands. This logic level intentionally identifies and prioritizes powered up and ready drives over their powered off neighbors on the same SMBus segment.<br>**Drive Functional** – Bit 5 is set to ‘1’ to indicate an NVM Subsystem is functional. If cleared to ‘0’, then there is an unrecoverable failure in the NVM Subsystem and the rest of the transmission may be invalid.<br>**Reset Not Required** - Bit 4 is set to ‘1’ to indicate the NVM Subsystem does not need a reset to resume normal operation. If cleared to ‘0’ then the NVM Subsystem has experienced an error that prevents continued normal operation. A Controller Level Reset is required to resume normal operation.<br>**Port 0 PCIe Link Active** - Bit 3 is set to ‘1’ to indicate the first port’s PCIe link is up (i.e., the Data Link Control and Management State Machine is in the DL_Active state). If cleared to ‘0’, then the PCIe link is down.<br>**Port 1 PCIe Link Active** - Bit 2 is set to ‘1’ to indicate the second port’s PCIe link is up. If cleared to ‘0’, then the second port’s PCIe link is down or not present.<br>Bits 1-0 shall be set to ‘1’. |

<table>
<tr><th>Command Code</th><th>Offset (byte)</th><th>Description</th></tr>
<tr><td rowspan="6"></td><td>02</td><td><b>SMART Warnings:</b> This field shall contain the Critical Warning field (byte 0) of the NVMe SMART / Health Information log. Each bit in this field shall be inverted from the NVMe definition (i.e., the management interface shall indicate a ‘0’ value while the corresponding bit is ‘1’ in the log page). Refer to the NVMe specification for bit definitions.<br><br>If there are multiple Controllers in the NVM Subsystem, the management endpoint shall combine the Critical Warning field from every Controller such that a bit in this field is:<ul><li>Cleared to ‘0’ if any Controller in the subsystem indicates a critical warning for that corresponding bit.</li><li>Set to ‘1’ if all Controllers in the NVM Subsystem do not indicate a critical warning for the corresponding bit.</li></ul></td></tr>
<tr><td>03</td><td><b>Composite Temperature (CTemp):</b> This field indicates the current temperature in degrees Celsius. If a temperature value is reported, it should be the same temperature as the Composite Temperature from the SMART log of hottest Controller in the NVM Subsystem. The reported temperature range is vendor specific, and shall not exceed the range -60 to +127°C. The 8 bit format of the data is shown below.<br><br>This field should not report a temperature when that is older than 5 seconds. If recent data is not available, the Management Endpoint should indicate a value of 80h for this field.<br><br><table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00h-7Eh</td><td>Temperature is measured in degrees Celsius (0 to 126C)</td></tr>
<tr><td>7Fh</td><td>127C or higher</td></tr>
<tr><td>80h</td><td>No temperature data or temperature data is more the 5 seconds old.</td></tr>
<tr><td>81h</td><td>Temperature sensor failure</td></tr>
<tr><td>82h-C3h</td><td>Reserved</td></tr>
<tr><td>C4</td><td>Temperature is -60C or lower</td></tr>
<tr><td>C5-FFh</td><td>Temperature measured in degrees Celsius is represented in two’s complement (-1 to -59C)</td></tr>
</table></td></tr>
<tr><td>04</td><td><b>Percentage Drive Life Used (PDLU):</b> Contains a vendor specific estimate of the percentage of NVM Subsystem NVM life used based on the actual usage and the manufacturer’s prediction of NVM life. If an NVM Subsystem has multiple Controllers the highest value is returned. A value of 100 indicates that the estimated endurance of the NVM in the NVM Subsystem has been consumed, but may not indicate an NVM Subsystem failure. The value is allowed to exceed 100. Percentages greater than 254 shall be represented as 255. This value should be updated once per power-on hour and equal the Percentage Used value in the NVMe SMART Health Log Page.</td></tr>
<tr><td>05:06</td><td>Reserved</td></tr>
<tr><td>07</td><td><b>PEC:</b> An 8 bit CRC calculated over the slave address, command code, second slave address and returned data. The algorithm is defined in the SMBus specification.</td></tr>
<tr></tr>
<tr><td rowspan="4">8</td><td>08</td><td><b>Length of identification:</b> Indicates number of additional bytes to read before encountering PEC. This value should always be 22 (16h) in implementations of this version of the spec.</td></tr>
<tr><td>10:09</td><td><b>Vendor ID:</b> The 2 byte vendor ID, assigned by the PCI SIG. Should match VID in the Identify Controller command response. Note the MSB is transmitted first.</td></tr>
<tr><td>11:30</td><td><b>Serial Number:</b> 20 characters that match the serial number in the NVMe Identify Controller command response. Note the first character is transmitted first.</td></tr>
<tr><td>31</td><td><b>PEC:</b> An 8 bit CRC calculated over the slave address, command code, second slave address and returned data. The algorithm is defined in the SMBus specification.</td></tr>
<tr><td>32+</td><td>32:255</td><td><b>Vendor Specific</b> – This data structure shall not exceed the maximum read length of 255 specified in the SMBus version 3 specification. Preferably length is not greater than 32 for compatibility with SMBus 2.0, additional blocks shall be on 8 byte boundaries.</td></tr>
</table>

# Appendix B – Example MCTP Messages & Message Integrity Check

Below are artificial MCTP Messages with their corresponding Message Integrity values. Figure 113 shows an example where the message is not an even number of Dwords and the MIC spans Dwords 7 and 8. The contents of the messages listed below should be used for reference and do not correspond to valid MCTP messages.

**Figure 111: MIC Example 1 – 32 Bytes of 0s**

| | **3** | **2** | **1** | **0** |
|---|---|---|---|---|
| **Dword 0** | 00h | 00h | 00h | 00h |
| **...** | ... | ... | ... | ... |
| **Dword 7** | 00h | 00h | 00h | 00h |
| **Dword 8 (MIC)** | **8Ah** | **91h** | **36h** | **AAh** |

**Figure 112: MIC Example 2 – 32 Bytes of 1s**

| | **3** | **2** | **1** | **0** |
|---|---|---|---|---|
| **Dword 0** | FFh | FFh | FFh | FFh |
| **...** | ... | ... | ... | ... |
| **Dword 7** | FFh | FFh | FFh | FFh |
| **Dword 8 (MIC)** | **62h** | **A8h** | **ABh** | **43h** |

**Figure 113: MIC Example 3 – 30 Incrementing Bytes from 0x00 to 0x1D**

| | **3** | **2** | **1** | **0** |
|---|---|---|---|---|
| **Dword 0** | 03h | 02h | 01h | 00h |
| **...** | ... | ... | ... | ... |
| **Dword 7 (MIC)** | **92h** | **D7h** | 1Dh | 1Ch |
| **Dword 8 (MIC)** | <unused> | | **1Eh** | **05h** |

**Figure 114: MIC Example 4 – 32 Decrementing Bytes from 0x1F to 0x00**

| | **3** | **2** | **1** | **0** |
|---|---|---|---|---|
| **Dword 0** | 1Ch | 1Dh | 1Eh | 1Fh |
| **...** | ... | ... | ... | ... |
| **Dword 7** | 03h | 02h | 01h | 00h |
| **Dword 8 (MIC)** | **11h** | **3Fh** | **DBh** | **5Ch** |