LEGAL NOTICE:

This NVM Express Management Interface revision 1.0a technical proposal is proprietary to the NVM Express, Inc. (also referred to as "Company") and/or its successors and assigns.

NOTICE TO USERS WHO ARE NVM EXPRESS, INC. MEMBERS: Members of NVM Express, Inc. have the right to use and implement this NVM Express Management Interface revision 1.0a technical proposal subject, however, to the Member's continued compliance with the Company's Intellectual Property Policy and Bylaws and the Member's Participation Agreement.

NOTICE TO NON-MEMBERS OF NVM EXPRESS, INC.: If you are not a Member of NVM Express, Inc. and you have obtained a copy of this document, you only have a right to review this document or make reference to or cite this document. Any such references or citations to this document must acknowledge NVM Express, Inc. copyright ownership of this document. The proper copyright citation or reference is as follows: **"© 2007 - 2018 NVM Express, Inc. ALL RIGHTS RESERVED."** When making any such citations or references to this document you are not permitted to revise, alter, modify, make any derivatives of, or otherwise amend the referenced portion of this document in any way without the prior express written permission of NVM Express, Inc. Nothing contained in this document shall be deemed as granting you any kind of license to implement or use this document or the specification described therein, or any of its contents, either expressly or impliedly, or to any intellectual property owned or controlled by NVM Express, Inc., including, without limitation, any trademarks of NVM Express, Inc.

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.

All product names, trademarks, registered trademarks, and/or servicemarks may be claimed as the property of their respective owners.

NVM Express Management Interface Workgroup
c/o NVM Express Administration
3855 SW 153rd Drive
Beaverton, OR 97003
admin@nvmexpress.org

NVM Express Technical Proposal for New Feature

Technical Proposal ID	**6002 – In-band NVMe-MI**
Change Date	**1/28/2018**
Builds on Specification	**NVM Express Management Interface 1.0a**

Technical Proposal Author(s)

Name	Company
Austin Bolen	Dell EMC

There are features that were added to NVMe-MI for out-of-band management that are also valuable in-band such as health monitoring at the FRU level and being able to read Vital Product Data. NVMe enclosure services is also targeted to be defined in the NVMe-MI spec and would also need a way to be implemented in-band. Adding in-band NVMe-MI support allows host operating systems and management applications to take advantage of these NVMe-MI features.

Revision History

Revision Date	Change Description
7/6/2017	• Initial draft.
9/12/2017	• Added diagrams and tables that illustrate how NVMe-MI Request/Response Messages map onto NVMe SQE/CQE respectively • Add requirement to make Request/Response Data Dword granular for the in-band tunneling mechanism • Removed Figure 35 changes since they are included in TP 001 (SES)
9/22/2017	• Changed spec that this TP builds on to 1.0a • Fixed acronyms for Response Data and Request Data • Updates to NVMe-MI Send/Receive command sections
10/2/2017	• Added command servicing diagrams and text to NVMe-MI Send/Receive command sections
10/9/2017	• More updates to NVMe-MI Send/Receive sections based on workgroup feedback
10/16/2017	• Noted in the definition of Control Primitives that they are only applicable in the out-of-band mechanism • Added “Packet Assembly into Messages” section with a few changes to standardize on “NVMe-MI Message” nomenclature • Added missing updates from VPD Write section
10/23/2017	• Removed FRU definition since it is part of TP 003. • Modified definition of NVMe-MI Messages to make it clear what type of MCTP Message they were • Noted in the definitions that Request Message and Response Messages apply in both in-band tunneling mechanism and out-of-band mechanism • Clarified that there is no size limit for NVMe-MI Messages using the in-band tunneling mechanism • Added definition for “NVMe Processing” and “Process” • Clarified that error checking is done in Process state • Added some missing updates • Editorial updates
10/30/2017	• Editorial updates • Added details in NVMe-MI Send/Receive servicing flow on how to set Tunneled NVMe Management Response
11/6/2017	• Accepted all changes
11/13/2017	• Fixed some reference numbers • Fixed cut-and-paste errors • Updated figures
1/28/2018	• Editorial changes to align with NVMe TP numbering, updated NVMe administration, and updated year.

Description of Specification Changes

Modify Section 1.1 (Overview) as shown below:

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) provides several mechanisms to manage NVMe Storage Devices.

One mechanism allows a Management Controller to communicate out-of-band with an NVMe Storage Device ~~NVM Subsystem~~ over one or more external interfaces. Another mechanism allows the NVMe-MI Management Interface Command Set to be tunneled in-band via the NVMe Admin commands NVMe-MI Send and NVMe-MI Receive. Refer to the NVM Express Specification and section 4.3 of this specification for additional details on the NVMe-MI Send and NVMe-MI Receive commands.

NVMe Storage Devices that comply with this specification are allowed to support only the out-of-band

mechanism, only the in-band tunneling mechanism, or both the out-of-band mechanism and in-band tunneling mechanism.

Since this specification builds on the NVM~~e~~ Express specification, knowledge of NVMe is assumed.

Modify Section 1.2 (Scope) as shown below:

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

NVMe-MI has the following key capabilities:

- Discover NVMe Storage ~~d~~Devices that are present and learn capabilities of each NVMe Storage ~~d~~Device
- Store data about the host environment enabling a Management Controller to query the data later
- Health and temperature monitoring
- Multiple ~~Command Slots~~outstanding commands to prevent a long latency command from blocking monitoring operations
- Out-of-band mechanism is host ~~P~~processor and operating system agnostic
- A standard format for VPD and defined mechanisms to read/write VPD contents
- Preserves data at rest security

Modify a portion of Section 1.2.1 (Outside of Scope) as shown below:

The management of NVMe ~~FRUs~~Storage Devices 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~~,~~ or PCIe switches~~ or management using any interface other than MCTP over PCIe VDM or SMBus/I2C~~. Co-ordination between multiple ~~Management Controllers~~Requesters or a ~~Management Controller~~Requester and a device other than a ~~Management Endpoint~~Responder is outside the scope of this specification. Refer to section 1.5.1 for the definitions of Requester and Responder.

Modify Section 1.3 (Theory of Operation) as shown below:

Commented [AB21]: TP 001 renamed this to "NVMe-MI Out-of-Band Protocol Layering"

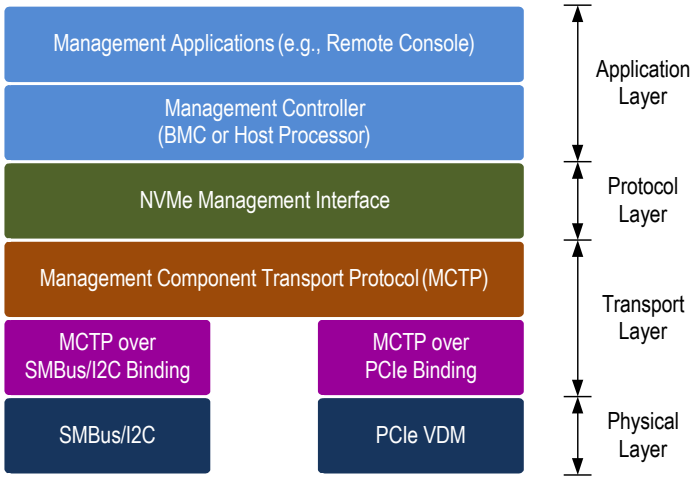
NVMe-MI is designed to provide a common interface over multiple physical layers (i.e., PCI Express, SMBus/I2C) for inventory, monitoring, configuration, and change management. The interface provides the flexibility necessary to manage NVMe Storage Devices ~~Subsystems~~ using an out-of-band mechanism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

1.3.1 Out-of-Band Theory of Operation

This specification defines two mechanisms for managing an NVMe Storage Device out-of-band. One is via the Management Component Transport Protocol (refer to section 1.3.1.1). The other is via a FRU Information Device (refer to section 1.3.1.2).

~~1.3.1~~1.3.1.1 Management Component Transport Protocol

Figure 1: NVMe Management Interface Out-of-Band Protocol Layering



For the out-of-band mechanism, 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~~Command Messages are submitted to one of two Command Slots associated with each Management Endpoint.

1.3.1.2 FRU Information Device

NVMe-MI also supports an out-of-band mechanism to access a FRU Information Device over SMBus/I2C as defined by the IPMI Platform Management FRU Information Storage Definition specification. The data stored in the FRU Information Device is referred to as Vital Product Data (refer to section 9.2).

1.3.2 In-Band Theory of Operation

For the in-band tunneling mechanism, NVMe-MI utilizes the NVMe Admin commands NVMe-MI Send and NVMe-MI Receive. Refer to the NVM Express Specification and section 4.3 of this specification for additional details on the NVMe-Send and NVMe-MI Receive commands.

Modify a portion of Section 1.4 (Architectural Model) as shown below:

Commented [AB22]: TP 001 renamed this to "NVMe Storage Device Architectural Model"

NVMe-MI supports an out-of-band mechanism and an in-band tunneling mechanism. The architectural model for NVMe Storage Devices that support the in-band tunneling mechanism follows the architectural model defined in the NVM Express specification. The architectural model defined by the NVM Express specification has been extended to support the out-of-band mechanism as described in section 1.4.1.

1.4.1 Out-of-Band Architectural Model Extensions

An NVMe storage device, such as a PCIe SSD, that implements the out-of-band mechanisms defined in this specification, consists of an NVM Subsystem with one or more Management Endpoints. There may be up to one Management Endpoint per PCIe port and up to one Management Endpoint per SMBus/I2C port. Each Management Endpoint 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.

Modify a portion of Section 1.5 (Conventions) as shown below:

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

Modify a portion of Section 1.5.1 (Definitions) as shown below:

1.5.1.a Control Primitive

A type of Request Message that may be sent while a Command Slot is processing a Command Message. A single packet ~~MCTP m~~NVMe-MI Message used to convey an NVMe-MI control request. Control Primitives are applicable only in the out-of-band mechanism and are prohibited in the in-band tunneling mechanism.

1.5.1.b Request Message

An ~~MCTP m~~NVMe-MI Message originating from a ~~Management Controller~~Requester. A Request Message may be a Command Message~~,~~ or a Control Primitive~~, or another type of MCTP message~~. Request Messages may be used in both the out-of-band mechanism and the in-band tunneling mechanism.

1.5.1.c Response Message

An ~~MCTP m~~NVMe-MI Message originating from a ~~Management Endpoint~~Responder in response to a Request Message. Response Messages may be used in both the out-of-band mechanism and the in-band tunneling mechanism.

1.5.1.d In-Band

Per the Management Component Transport Protocol (MCTP) Overview White Paper, in-band management is management that operates with the support of hardware components that are critical to and used by the operating system. The in-band communication path for NVMe-MI is via the NVMe Admin Queue using the NVMe-MI Send and NVMe-MI Receive commands from host software to an NVMe Controller. Refer to the NVM Express Specification and section 4.3 of this specification for additional details on the NVMe-Send and NVMe-MI Receive commands.

1.5.1.e Out-of-Band

Per the Management Component Transport Protocol (MCTP) Overview White Paper, out-of-band management is management that operates with hardware resources and components that are independent of the operating system's control. The out-of-band communication paths for NVMe-MI are via MCTP over SMBus/I2C or MCTP over PCIe VDM from a Management Controller to a Management Endpoint or via the access mechanism defined by the IPMI Platform Management FRU Information Storage Definition specification for accessing a FRU Information Device from a Management Controller over SMBus/I2C.

1.5.1.f NVMe-MI Message

A type of MCTP Message that is defined by this specification in sections 3.1 and 4.1. See the MCTP IDs and Codes specification and the NVMe Management Messages over MCTP Binding Specification for more details on this type of MCTP Message (note that NVMe-MI Messages are referred to as NVM Express Management Messages over MCTP in these specifications).

1.5.1.g Requester

The entity that sends Request Messages and receives Response Messages. For the out-of-band mechanism, the Requester is a Management Controller. For the in-band mechanism, the Requester is host software.

1.5.1.h Responder

The entity that receives Request Messages and sends back Response Messages. For the out-of-band mechanism, the Responder is a Management Endpoint. For the in-band mechanism, the Responder is an NVMe Controller.

1.5.1.i NVMe Processing

NVMe command processing as defined by the NVM Express specification. The term NVMe Processing is used to distinguish command processing as defined by the NVM Express specification from the Command Message processing defined by this specification (refer to 1.5.1.j).

1.5.1.j Process

This is the state when a Command Message is processed. Processing of a Command Message consists of checking for errors with the Command Message and performing the actions specified by the Command Message. This state is applicable in both the out-of-band mechanism and the in-band tunneling mechanism. Refer to section 4.2 for additional details on the Process state in the out-of-band mechanism. Refer to section 4.3 for additional details on the Process state in the in-band tunneling mechanism.

This specification uses the terms process/processing/processed to refer to actions performed in the Process state. These terms are distinct from the NVMe Processing term used to describe NVMe command processing as defined by the NVM Express specification (refer to 1.5.1.i).

Modify a portion of Section 1.7 (References) as shown below:

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

MCTP Overview White Paper (DSP2016), version 1.0.0. Available from http://dmtf.org.

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

Modify Section 2.1 (PCI Express) as shown below:

PCI Express is used as the physical layer in both the out-of-band mechanism and the in-band tunneling mechanism in NVMe-MI.

For the out-of-band mechanism, ~~A~~a PCIe port in an NVMe Storage Device ~~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.

For the in-band tunneling mechanism, host software issues NVMe Admin commands (NVMe-MI Send and NVMe-MI Receive) to the NVMe Admin Queue over PCI Express. Refer to the NVM Express Specification and section 4.3 of this specification for additional details on the NVMe-MI Send and NVMe-MI Receive commands.

Modify a portion of Section 2.2 (SMBus/I2C) as shown below:

The SMBus/I2C physical layer is only applicable in the out-of-band mechanism. If the NVMe Storage Device ~~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.

Modify a portion of Section 3.0 (Message Transport) as shown below:

~~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).~~

NVMe-MI is supported on multiple message transports. The message format is the same for the out-of-band mechanism and the in-band tunneling mechanism and is described in section 3.1. The out-of-band message transport is described in section 3.2. The in-band tunneling message transport is described in section 3.3.

Move section 3.2 (MCTP Messages) and all subsections to Section 3.1

Modify a portion of the old Section 3.2 (MCTP Messages) as shown below:

3.1 ~~MCTP~~NVMe-MI Messages

NVMe-MI Messages, which are a type of MCTP message, are used in both the out-of-band mechanism and the in-band tunneling mechanism. The format of an NVMe-MI Message is shown in **Error! Reference source not found.**Figure 11.

In the out-of-band mechanism, a~~A~~n ~~MCTP m~~NVMe-MI Message consists of the payload of one or more MCTP packets. The maximum sized NVMe-MI M~~m~~essage is 4224 bytes (4K + 128). Refer to the NVMe Management Messages over MCTP Binding Specification. NVMe-MI Messages with lengths greater than 4224 are considered invalid NVMe-MI ~~m~~Messages. ~~The format of an NVMe-MI Message is shown in **Error! Reference source not found.**Figure 11.~~ See section 4.2 for details on how NVMe-MI Messages are used in the out-of-band mechanism.

In the in-band tunneling mechanism, NVMe-MI Messages do not consist of MCTP packets and there is no defined maximum sized NVMe-MI Message. See section 4.3 for details on how NVMe-MI Messages are used in the in-band tunneling mechanism.

Rename Figure 10 as shown below:

Figure 10: NVMe-MI M~~CTP M~~essage

Modify old Section 3.2.1 (Message Fields) as follows:

The format of an NVMe-MI ~~m~~Message consists of a Message Header in the first Dword, followed by the Message Data~~., and~~ If the Integrity Check (IC) field is set to '1' then the NVMe-MI Message 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 Messages (refer to the MCTP IDs and Codes specification). The Integrity Check (IC) field indicates whether the NVMe-MI ~~m~~Message is ~~covered~~ protected by a~~n overall MCTP~~ Message Integrity Check. All NVMe-MI ~~m~~Messages in the out-of-band mechanism are protected by a 32-bit CRC computed over the ~~m~~Message ~~b~~Body contents. The IC field shall be set to '1' in all NVMe-MI M~~CTP m~~essages in the out-of-band mechanism.

Commented [AB23]: Figure 9 after sections are moved.

Commented [AB24]: Figure 9 after sections are moved.

Commented [AB25]: Figure 10 after the sections are moved.

The Integrity Check (IC) field shall be cleared to '0'~~zero~~ in all NVMe-MI Messages in the in-band tunneling mechanism.

The Request or Response (ROR) bit in the Message Header specifies whether the NVMe-MI ~~MCTP m~~Message is associated with a Request Message or a Response Message. The NVMe-MI 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 NVMe-MI ~~m~~Message is associated in the out-of-band mechanism. Refer to section 4 for additional information about Command Slots.

Modify Figure 11: NVMe-MI MCTP Message Fields as follows:

Commented [BA6]: Still Figure 14 after the sections are moved.

Commented [AB27]: Section 4.2 after the sections are moved.

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

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>0</td><td>MCTP Data (MCTPD): This field contains the Message Type and Integrity Check fields as defined by the MCTP Base Specification.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7</td><td>Integrity Check (IC): This field is defined by the MCTP Base Specification and indicates whether the MCTP message is covered by an overall MCTP Message Integrity Check. All NVMe-MI ~~m~~Messages in the out-of-band mechanism are protected by a CRC and thus this bit shall be set to ‘1’ in all out-of-band NVMe-MI ~~m~~Messages.

All NVMe-MI Messages in the in-band tunneling mechanism are not protected by a CRC and thus this bit shall be cleareMd to ‘0’ in all in-band NVMe-MI Messages.</td></tr>
<tr><td>6:0</td><td>Message Type (MT): This field is defined by the MCTP Base Specification for the message type. This field shall be set to 4h in all NVMe-MI ~~m~~Messages. Refer to MCTP IDs and Codes .</td></tr>
</table></td></tr>
<tr><td>1</td><td>NVMe-MI Message Parameters (NMP): This field contains parameters applicable to the NVMe-MI Message.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7</td><td>Request or Response (ROR): 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>NVMe-MI Message Type (NMIMT): This field specifies the NVMe-MI Message Type. Refer to the sections referenced in the table below for details about each NVMe-MI Message Type and whether they apply to the out-of-band mechanism, the in-band tunneling mechanism, or both.
<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>Command Slot Identifier (CSI): This field indicates the Command Slot with which the NVMe-MI M~~m~~essage 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. This field is only applicable to NVMe-MI Messages in the out-of-band mechanism. This field is unused and shall be set to 0h for NVMe-MI Messages in the in-band tunneling mechanism.
<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>Message Data (DATA): This field contains the NVMe-MI ~~m~~Message payload. The format of this field depends on the NVMe-MI Message Type.</td></tr>
<tr><td>x+3:x</td><td>Message Integrity Check (MIC): If the Integrety Check (IC) field is set ‘1’, then ~~T~~this field contains a CRC computed over the contents of the NVMe-MI ~~m~~Message. Refer to section 3.2.1.1.

If the Integrety Check (IC) field is cleared to ‘0’, then this field is not included in the NVMe-MI Message.</td></tr>
</table>

Modify a portion of the old Section 3.2.1.1 (Message Integrity Check) as shown below:

If the Integrity Check (IC) field is set to ‘1’, ~~T~~then the Message Integrity Check field contains a 32-bit CRC computed over the contents of the NVMe-MI M~~CTP m~~essage.

Modify a portion of the old Section 3.2.1.1 (Message Integrity Check) as shown below:

Commented [BA8]: Figure 9 after the section are moved.

Commented [AB29]: Section 4.2.1 after sections are moved.

Commented [AB210]: Section 3.1.1.1 after sections are moved.

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

Modify a portion of the old Section 3.2.1.1 (Message Integrity Check) as shown below:
See Section 4.4.5 for special requirements on how to construct the ~~NVMe-MI~~ Response Message when the Management Controller issues a Replay of a Response Message with a non-zero Response Replay Offset.

Commented [AB211]: Section 4.2.1.5 after section are moved.

Create a new Section 3.2 (Out-of-Band Message Transport):

3.2 Out-of-Band Message Transport

The out-of-band mechanism in 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 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).

Move old Section 3.1 (MCTP Packet) to be under new section 3.2 (Out-of-Band Message Transport) with the following modifications:

~~3.1~~3.2.1 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. In NVMe-MI, the MCTP messages are referred to as NVMe-MI Messages (refer to section 1.5.1.f). Refer to section 3.2.2 for details on how MCTP packets are assembled into NVMe-MI Messages. 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.

Commented [BA12]: Packet Assembly into Messages is section 3.2.1.1 after sections are moved

Commented [BA13]: Figure 11 after sections are moved

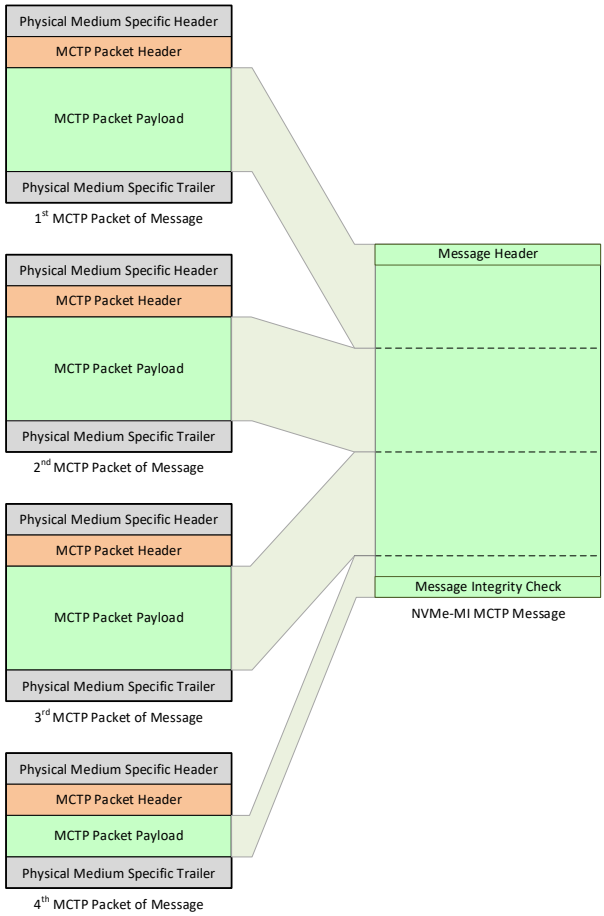
Move old Section 3.2.2 (Packet Assembly into Messages) to be under new section 3.2.1 (MCTP Packet) with the following modifications:

~~3.2.2~~3.2.1.1 Packet Assembly into Messages

An NVMe-MI Message may be split into multiple MCTP Packet Payloads and sent as a series of packets. An example NVMe-MI M~~m~~essage 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.

Commented [BA14]: Still Figure 13 after sections are moved

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



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 M~~CTP m~~essage has been assembled, the Message Integrity Check is verified. If the Message Integrity Check passes, then the NVMe-MI ~~m~~Message is processed. If the Message Integrity Check fails, then the NVMe-MI ~~m~~Message is discarded. Refer to 4.3.

Commented [BA15]: Still Figure 13 after sections are moved

Commented [BA16]: Section 4.2 after sections are moved.

Indent old Section 3.3 (Error Handling) to be under new section 3.2 (Out-of-Band Message Transport) and rename section to Out-of-Band Error Handling:

~~3.~~3.2.2 Out-of-Band Error Handling

Add the following new section after old Section 3.3 (Error Handling):

3.3 In-Band Tunneling Message Transport

The in-band tunneling mechanism in NVMe-MI utilizes the NVMe Admin commands NVMe-MI Send and NVMe-MI Receive as a message transport. Refer to the NVM Express Specification and section 4.3 of this specification for additional details on the NVMe-Send and NVMe-MI Receive commands.

Modify Section 4 (Message Processing Model) as shown below:

4 Message Servicing ~~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).~~
NVMe-MI Messages are used for communication in both the out-of-band and in-band tunneling message servicing model and are described in section 4.1. NVMe-MI supports multiple message servicing models. The out-of-band message servicing model is described in section 4.2. The in-band tunneling message servicing model is described in section 4.3.

4.1 NVMe-MI ~~MCTP~~ Messages

Figure 14 illustrates the taxonomy of NVMe-MI M~~CTP m~~essages. The two main categories of NVMe-MI Messages are Request Messages and Response Messages. Request Messages are sent by a Management Controller to a Management Endpoint when using the out-of-band mechanism. Request Messages are sent by host software to an NVMe Controller when using the in-band tunneling mechanism. The entity sending the Request Message is collectively referred to as the Requester and the entity receiving the Request Message is collectively referred to as the Responder. After receiving a Request Message, the Responder will process the Request Message. When processing is complete, the Responder sends a Response Message back to the Requester.

A Request Message may be classified as a ~~c~~Command Message or a Control Primitive. Command~~s~~ Messages specify an operation to be performed by the ~~Management Endpoint~~Responder and may be further classified as an NVMe-MI ~~c~~Command, an NVMe Admin ~~c~~Command, or a PCIe ~~c~~Command. Control Primitives are used in the out-of-band mechanism to affect the servicing~~processing~~ of a previously issued Command Message or get the state of a Command Slot and Management Endpoint ~~and are described in~~ (refer to Section 4.4).

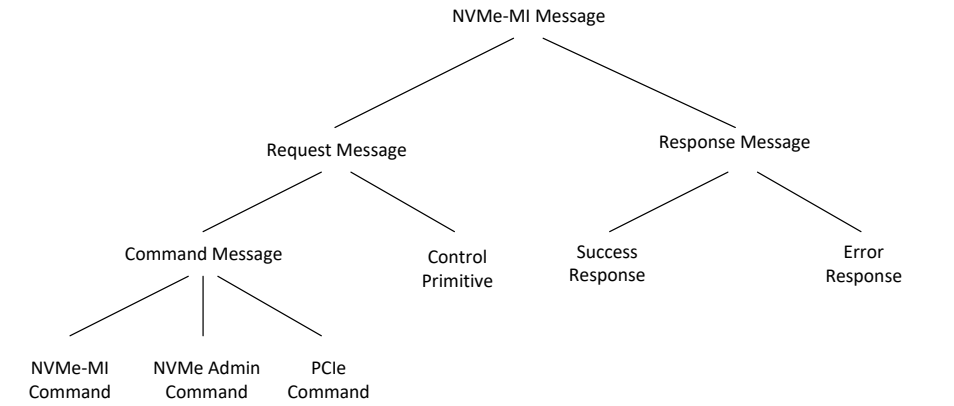
~~Unlike other NVMe-MI MCTP messages that may span multiple MCTP packets, messages containing a Control Primitive shall consist of exactly one MCTP packet.~~

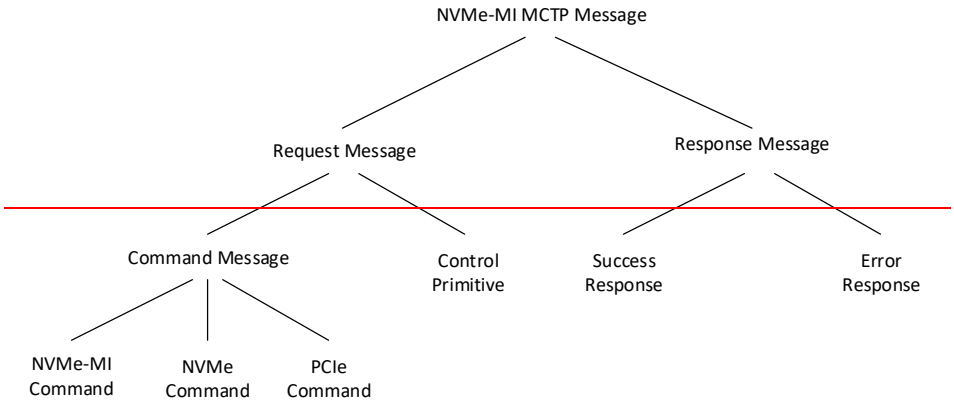
A Response Message may be classified as a ~~s~~Success ~~r~~Response or an ~~e~~Error ~~r~~Response.

Modify Figure 14 (NVMe-MI MCTP Message Taxonomy) as shown below:

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

Commented [BA17]: Section 4.2.1 after section are moved





Indent old Section 4.1 (Request Messages) to be under new section 4.1 (NVMe-MI Messages) and modify as shown below:

~~4.1~~4.1.1 Request Messages

Request Messages are NVMe-MI ~~m~~Messages that are generated by a Requester~~Management Controller~~ to send to a ~~Responder~~Management Endpoint.
Request Messages specify an action to be performed by the ~~Responder~~Management Endpoint. Request Messages are either Control Primitives (refer to 4.4) or Command Messages. The format of the ~~m~~Message ~~b~~Body for a Command Message is command set specific and is specified by the NMIMT field in the ~~m~~Message ~~h~~Header.
The NVMe Management Interface supports three command sets:

- The Management Interface ~~c~~Command ~~s~~Set is described in chapter 5.
- The NVM Express Admin ~~c~~Command ~~s~~Set is described in chapter 6.
- The PCIe ~~c~~Command ~~s~~Set is described in chapter 7.

Commented [AB218]: Section 4.2.1 after section are moved

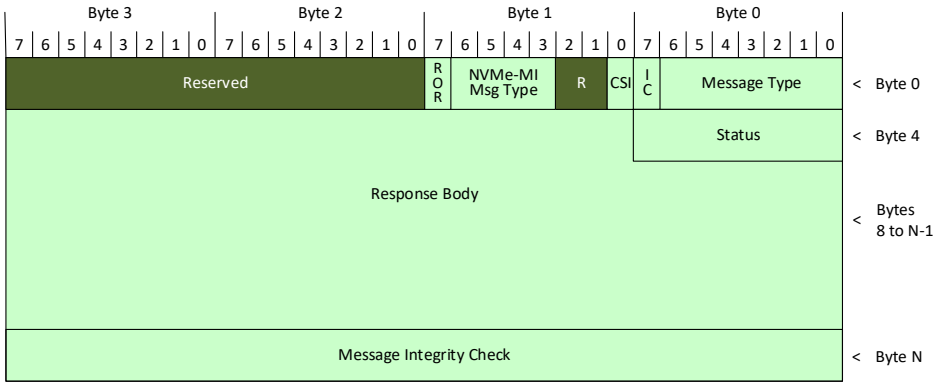
Indent old Section 4.2 (Response Messages) to be under new section 4.1 (NVMe-MI Messages) and modify as shown below:

~~4.14~~.1.2 Response Messages

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

The format of a Response Message is shown in Figure and Figure. The first Dword contains the ~~NVMe-MI m~~Message ~~h~~Header. The Status field encodes the status associated with the Response Message. This is followed by the Response Body whose format is NVMe-MI Message Type and ~~r~~Response Message ~~s~~Status specific. Finally, if the Integrity Check (IC) field is set to '1' then the Response Message ends with the NVMe-MI Message Integrity Check field.

Figure 15: Response Message Format



In the out-of-band mechanism, ~~T~~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.

In the in-band tunneling mechanism, the CSI field in the Message Header is reserved and shall be set to a value of '1'.

Figure 16: Response Message Fields

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

Commented [AB219]: Section 3.1 after sections are moved.

Commented [AB220]: Section 3.1 after sections are moved.

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 ~~s~~Success ~~r~~Response. The format of the ~~r~~Response ~~b~~Body for a ~~s~~Success ~~r~~Response is dependent on the NVMe-MI ~~m~~Message ~~t~~Type and is described later in this specifiation.

A Response Message Status value other than~~t~~ Success indicates that an error occurred during servicing~~processing~~ of the corresponding ~~command~~Request Message and that the ~~r~~Response Message

is an ~~eE~~rror ~~rR~~esponse. The format of the ~~rR~~esponse ~~bB~~ody is dependent on the Response Message Status value as shown in Figure 17. If multiple errors are present, a ~~Management Endpoint~~Responder 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 ~~c~~Command Message is in progress and requires more time to complete processing. When this Response Message Status value is used in a Response Message, a subsequent Response ~~m~~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~~Request Message could not be executed due to a vendor specific internal error.	Refer to 4.2.1
03h	**Invalid Command Opcode:** The associated command opcode field is not valid. Invalid opcodes include reserved and optional opcodes that are not implemented.	Refer to 4.2.2
04h	**Invalid Parameter:** Invalid ~~command~~ parameter field value. Request Messages received with reserved values in defined fields shall be completed with an Invalid Parameter Error Response Message. Request Messages received with reserved or unimplemented values in defined fields shall be completed with an Invalid Parameter Error Response Message. Other error conditions that result in Invalid Parameter Error Response Message are noted elsewhere in this specification.	Refer to 4.2.1
05h	**Invalid Command Size:** The ~~Command~~ Message ~~bB~~ody of the Command Message was larger or smaller than that expected ~~by the command~~ due to a reason other than too much or too little Request~~input d~~Data (e.g., the ~~c~~Command Message did not contain all the required parameters or no ~~input~~Request ~~d~~Data was expected but the ~~command message body~~Request Data is larger than that needed to contain the required parameters). The expected ~~command m~~Message ~~bB~~ody size is determined by the ~~command~~NVMe-MI Message Type and opcode assuming no other errors are detected (e.g., Invalid Command Opcode or Invalid Parameter).	Refer to 4.2.1
06h	**Invalid Command Input Data Size:** The Command Message requires ~~input d~~Request Data and contains too much or too little ~~input d~~Request Data.	Refer to 4.2.1
07h	**Access Denied:** A ~~command~~ Request Message was prohibited from being ~~executed~~ processed 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

Commented [AB221]: Section 4.1.2.1 after the sections are moved.

Commented [AB222]: Section 4.1.2.1 after the sections are moved.

Commented [AB223]: Section 4.1.2.2 after the sections are moved.

Commented [AB224]: Section 4.1.2.2 after the sections are moved.

Commented [AB225]: Section 4.1.2.1 after the sections are moved.

Commented [AB226]: Section 4.1.2.1 after the sections are moved.

Commented [AB227]: Section 4.1.2.1 after the sections are moved.

Commented [AB228]: Section 4.1.2.1 after the sections are moved.

Commented [AB229]: Section 4.1.2.1 after the sections are moved.

Commented [AB230]: Section 4.1.2.1 after the sections are moved.

Indent old Section 4.2.1 (Generic Error Response) to be under new section 4.1 (NVMe-MI Messages) and modify as shown below:

~~4.2.14~~.1.2.1 Generic Error Response

A ~~g~~Generic ~~e~~Error ~~r~~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 ~~g~~Generic ~~e~~Error ~~r~~Response is shown in Figure 18.

Indent old Section 4.2.1 (Invalid Parameter Error Response) to be under new section 4.1 (NVMe-MI Messages) and modify as shown below:

~~4.2.14~~.1.2.2 Invalid Parameter Error Response

An ~~i~~Invalid ~~p~~Parameter ~~e~~Error ~~r~~Response is generated for error responses where the Status field is set to Invalid Parameter. The format of an ~~i~~Invalid ~~p~~Parameter ~~e~~Error ~~r~~Response is shown in Figure 19 and the response specific fields are summarized in Figure 20.

Modify old Section 4.3 (Command Processing Model) as shown below:

~~4.34~~.2 Out-of-Band Message Servicing~~Command Processing~~ Model

The out-of-band mechanism in NVMe-MI utilizes a request and response servicing 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).

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

NVMe-MI utilizes Command Slots for ~~c~~Command Message servicing. A Management Controller should not send a new Command Message to a Command Slot until the Response Message for the previously issued ~~c~~Command Message to that Command Slot has been received. Each Management Endpoint contains ~~2~~two Command Slots that each include state information and a Pause flag (refer to 4.4.4).

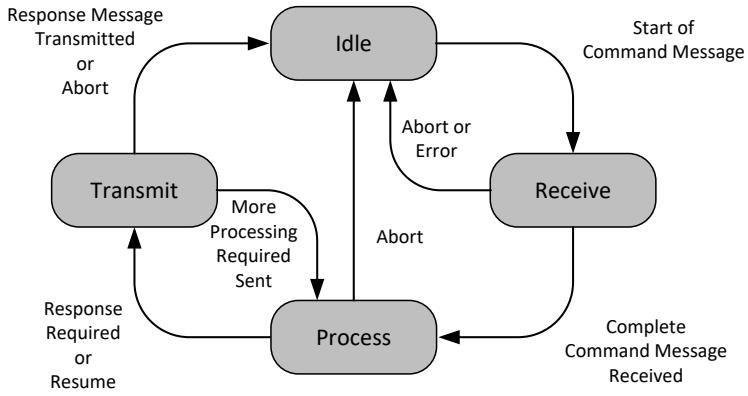
Commented [AB231]: Section 4.2.1.4 after sections are moved.

A Management Controller sends a Command Message to a Management Endpoint that targets a specific Command Slot in the Management Endpoint. The Management Endpoint assembles MCTP packets into Command Messages targeting a Command Slot. The Command Slot remains allocated to the Command Message until servicing of the Command Message has completed and command servicing transitions back to the Idle state.

A Command Message is the only type of multi-packet ~~MCTP~~ NVMe-MI ~~m~~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~~two independent streams of Command Messages to a Management Endpoint. The Command Message associated with each Command Slot are serviced~~processed~~ in parallel. If the NVM Subsystem implements multiple Management Endpoints, then command servicing of each Management Endpoint occurs in parallel. A NVM Subsystem that implements *N* Management Endpoints may have up to 2*N* Command Messages serviced in parallel.

The Command Servicing State Diagram in Figure 21 is used to describe functional requirements and does not mandate an implementation.

Figure 21: Command Servicing State Diagram



1. **Idle**: This is the default state of the command servicing state machine (e.g., following a reset). Command servicing transitions from Idle to the Receive state when the first MCTP packet of a ~~MCTP NVMe-MI cC~~ommand ~~mM~~essage is received (i.e., an MCTP packet with the SOM bit in the MCTP packet header set to '1' and the Message Type set to 4h).

2. **Receive**: The state when the first packet of a Command Message has been received and the message is being assembled and/or validated. Command servicing 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). Command servicing 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 ~~cC~~ommand Message consists of checking for errors with the Command Message and performing the actions specified by the ~~cC~~ommand Message or aborting the ~~cC~~ommand Message. Command servicing transitions from Process to the Transmit state when a ~~rR~~esponse Message is required to be sent (i.e., the Pause Flag is cleared to '0' and either of the following are true: all processing of the ~~cC~~ommand Message has completed or command processing is expected to exceed the corresponding transport binding specification response timeout). Command servicing 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. Command servicing transitions from the Transmit to the Idle state once the entire NVMe-MI ~~mM~~essage associated with the response to the ~~cC~~ommand Message has been transmitted on the physical medium or due to an Abort Control Primitive (refer to 4.4.3). If command servicing did not complete in the Process state, then the Management Endpoint transmits a ~~rR~~esponse Message with status More Processing Required and the command servicing transitions back to the Process state.

The behavior of receiving two or more overlapping Command Messages to the same Command Slot is undefined. If this results in the Management Endpoint discarding a Command Message, then this is considered receiving a Command Message to a non-Idle Command Slot (CMNICS). Refer to section 4.4.4.

Commented [AB232]: Section 3.2.1.1 after sections are moved.

Commented [AB233]: Section 3.1.1.1 after sections are moved.

Commented [AB234]: Section 4.2.1.3 after sections are moved.

Commented [AB235]: Section 4.2.1.3 after sections are moved.

Commented [AB236]: Section 4.2.1.4 after sections are moved.

Indent old Section 4.4 (Control Primitives) to be under new section 4.2 (Out-of-Band Message Servicing Model) and modify as shown below:

~~4.4~~4.2.1 Control Primitives

Control Primitives are Request Messages sent from a Management Controller to a Management Endpoint to affect the servicing of a previously issued Command Message or get the state of a Command Slot and Management Endpoint~~command processing flow~~ and are applicable only in the out-of-band mechanism and are prohibited in the in-band tunneling mechanism. 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

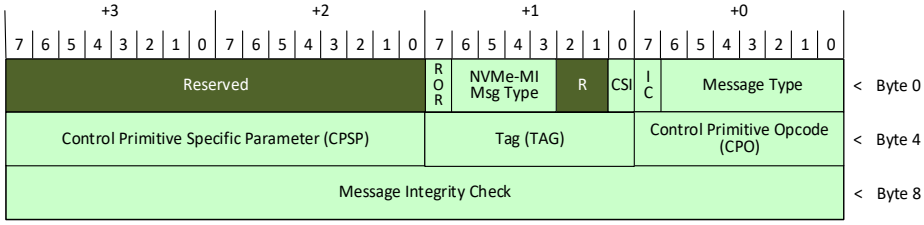


Figure 23: Control Primitive ~~Request Message~~ Fields

Byte	Description
03:00	**NVMe-MI Message Header (NMH)**: 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 ~~r~~Response Message. 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.

Commented [AB237]: Section 3.1 after sections are moved.

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
F0h - FFh	O	Vendor Specific

NOTES:

1. Optional or Mandatory; O – optional and M - mandatory

The format of a ~~s~~Success ~~r~~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

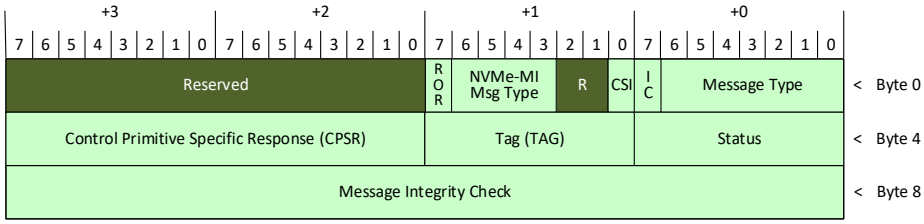


Figure 26: Control Primitive Success Response ~~Message~~ Fields

Byte	Description
03:00	**NVMe-MI Message Header (NMH)**: Refer to Section 3.2.
04	**Status (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.

Commented [AB238]: Section 3.1 after sections are moved.

NOTE!!!! Most of section 3.2 got moved to 3.1… but 3.2.2 got moved to 3.2.1.1. Need to review all references to section 3.2 to see if they need to be changed to 3.1 or 3.2.1.1. Most should go to 3.1.

Commented [AB239]: Section 4.1.2 after sections are moved.

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.

~~4.4.14~~.2.1.1 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

Byte	Description
07:06	**Control Primitive Specific Response (CPSR):** This field is used to return Control Primitive specific status.

Bits	Description
15:02	Reserved
01	**Pause Flag Status Slot 1 (PFSS1):** 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.
00	**Pause Flag Status Slot 0 (PFSS0):** 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.

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.

~~4.4.1~~4.2.1.2 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:

Commented [AB240]: Section 4.2.1.4 after sections are moved.

Commented [AB241]: Section 4.2.1.4 after sections are moved.

Commented [AB242]: Section 4.2.1.4 after sections are moved.

Commented [AB243]: Section 4.2.1.4 after sections are moved.

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.

Commented [AB244]: Section 4.2.1.4 after sections are moved.

Commented [AB245]: Section 4.2.1.4 after sections are moved.

Commented [AB246]: Section 4.2.1.4 after sections are moved.

~~4.4.1~~4.2.1.3 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 servicing~~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 servicing~~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 2**.

Figure 2: Abort Control Primitive Success Response Message Fields

Byte	Description
07:06	**Control Primitive Specific Response (CPSR):** This field is used to return Control Primitive specific status. **Bits** / **Description** 15:02 – Reserved 01:00 – **Command Processing Abort Status (CPAS):** This field indicates the effect of the Abort primitive on the processing of the Command Message associated with the Command Slot. 0h – Command aborted after processing completed or no command to abort. 1h – Command aborted before processing began 2h – Command processing partially completed. 3h – Reserved

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.

~~4.4.1~~4.2.1.4 Get State

The Get State Control Primitive is used to ~~check~~get the state of a Command Slot and Management Endpoint.

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
07:06	**Control Primitive Specific Parameter (CPSP):** This field is used to to pass Control Primitive specific parameter information. **Bits** / **Description** 15:01 – Reserved 00 – **Clear Error State Flags (CESF):** This field specifies whether or not to clear the error state flags when completing this command.

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**.

Figure 30: Get State Control Primitive Success Response Message Fields

Byte	Description
07:06	**Control Primitive Specific Response (CPSR):** This field is used to return Control Primitive specific status. **Bits** / **CS Specific[1]** / **Description**

15	Yes	**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. 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	No	**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	No	**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	No	**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	No	**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	No	**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	No	**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	No	**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	No	**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	No	**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	No	**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	No	**Bad Message Integrity Check Error (BMICE):** 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	No	**Command Message to non-Idle Command Slot (CMNICS):** This field is set to '1' if the Management Endpoint discarded one or more Command Messages due to overlapping Command Messages to a Command Slot since the last time Get State primitive was executed with the CESF field set to '1'.
02		Reserved
01:00	Yes	**Slot Command Servicing State (SSTA):** This field indicates the current command servicing state of the Command Slot. An implementation may choose to indicate only the Idle and Process states in this field. Refer to Figure 21. **Value** / **Description** 0h – Idle 1h – Receive 2h – Process 3h – Transmit
Notes: **1.** Command Slot Specific. Yes in this column indicates the value of the field within a Management Endpoint is independent per Command Slot.		

~~4.4.14~~.2.1.5 Replay

The Replay Control Primitive is used to retransmit the Response Message for the last Command Message processed in a Command Slot. The replayed Response Message forms a new MCTP Response Message with Message Data starting from Response Replay Offset of the original Response Message and continuing to the end of the Response Message, including the original MIC. The first packet shall have SOM set and shall include the Message Header of the original Response Message even if the Response Replay Offset is not zero.

Note that the Management Controller will need extensions to the MCTP Base Specification in its MCTP layer in order to Replay a Response Message using a non-zero Response Replay Offset. No extensions to the MCTP Base Specification are needed to Replay with Response Replay Offset equal to zero. For the case where a Management Controller chooses to use a non-zero Response Replay Offset, the MCTP Base Specification requires terminating message assembly for certain errors (i.e. receiving a packet with bad packet data integrity). If a Management Controller receives a number of packets with no errors in a Response Message and then gets an error on a packet that causes termination of message assembly, the Management Controller will need extensions in its MCTP layer to forward the packets it received with no errors to its NVMe-MI layer prior to terminating message assembly. The Management Controller can then issue a Replay to get the second part of the Response Message using a non-zero Response Replay Offset. The Management Controller's NVMe-MI layer can then assemble the two partial Response Messages to create the whole Response Message. The MIC can then be validated across the whole Response Message as described in Section 3.2.1.1.

Commented [AB247]: Section 3.1.1.1 after sections are moved.

The format of the CPSP field in the Control Primitive Request Message is shown in **Figure 31**.

Figure 31: Replay Control Primitive Request Message Fields

Byte	Description

07:06	**Control Primitive Specific Parameter (CPSP):** This field is used to to pass Control Primitive specific parameter information. **Bits** / **Description** 15:08 — Reserved 07:00 — **Response Replay Offset (RRO):** 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. 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. 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.		

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

Byte	Description
07:06	**Control Primitive Specific Response (CPSR):** This field is used to return Control Primitive specific status. **Bit** / **Description** 15:01 — Reserved 00 — **Response Replay (RR):** This bit indicates if a previous Response Message is retransmitted. This field is set to '1' if the requested Response Message is retransmitted by the Management Endpoint. This field is cleared to '0' if the requested Response Message is not retransmitted.

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 **Error! Reference source not found.**) 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 and then transmits a Response Message with success status with the RR field set to '1'. The Management Endpoint transmits a Response Message containing the packets starting at the packet offset specified in the Response Replay Offset field of the Replay 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.

Indent old Section 4.5 (Error Handling) to be under new section 4.2 (Out-of-Band Message Servicing Model) and modify as shown below:

4.54.2.2 Out-of-Band Error Handling

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

4.54.2.2.1 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 ~~c~~Command~~s~~ Messages that take less than or the required time to complete.

4.54.2.2.2 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.

Add new Section 4.3 as shown below:

4.3 In-Band Tunneling Message Servicing Model

The in-band tunneling mechanism in NVMe-MI utilizes two NVMe Admin commands (NVMe-MI Send and NVMe-MI Receive). NVMe-MI Send is used to tunnel an NVMe-MI Command from host software to an NVMe Controller that transfers data from the host to the NVMe Controller (similar to a write operation).

NVMe-MI Receive is used to tunnel an NVMe-MI Command from a host to an NVMe Controller that transfers data from the NVMe Controller to the host (similar to a read operation).

Refer to the NVM Express Specification for additional details on the NVMe-Send and NVMe-MI Receive commands. Additional details on NVMe-MI Send are in section 4.3.1 and additional details on NVMe-MI Receive are in section 4.3.2.

4.3.1 NVMe-MI Send Command

The NVMe-MI Send command is an NVMe Admin Command as defined by this specification and the NVM Express Specification. It is used to tunnel an NVMe-MI Command in-band from host software to an NVMe Controller that transfers data from a host to an NVMe Controller (similar to a write operation). The data being transferred is in one or more of the following locations: Request Data, NVMe Management Dword 0, NVMe Management Dword 1.

The mapping of how NVMe-MI Commands are tunneled inside of NVMe-MI Send commands is described in section 4.3.1.1. The NVMe-MI Send command servicing model is described in section 4.3.1.2.

4.3.1.1 NVMe-MI Send Command Request Message to NVMe Admin Command SQE Mapping

In order to tunnel NVMe-MI Commands in-band via NVMe-MI Send, NVMe-MI Request Messages are mapped onto NVMe Submission Queue Entries (SQE) as shown pictorially in Figure 4.3.1.1_1 and in table form in Figure 4.3.1.1_2. NVMe-MI Response Messages are mapped on to NVMe Completion Queue Entries (CQE) as shown pictorially in Figure 4.3.1.1_3 and in table form in Figure 4.3.1.1_4. Refer to the NVM Express specification for details on NVMe Submission Queue Entries and NVMe Completion Queue Entries.

Figure 4.3.1.1_1 NVMe-MI Send Command Request Message to NVMe Admin Command SQE Mapping Diagram

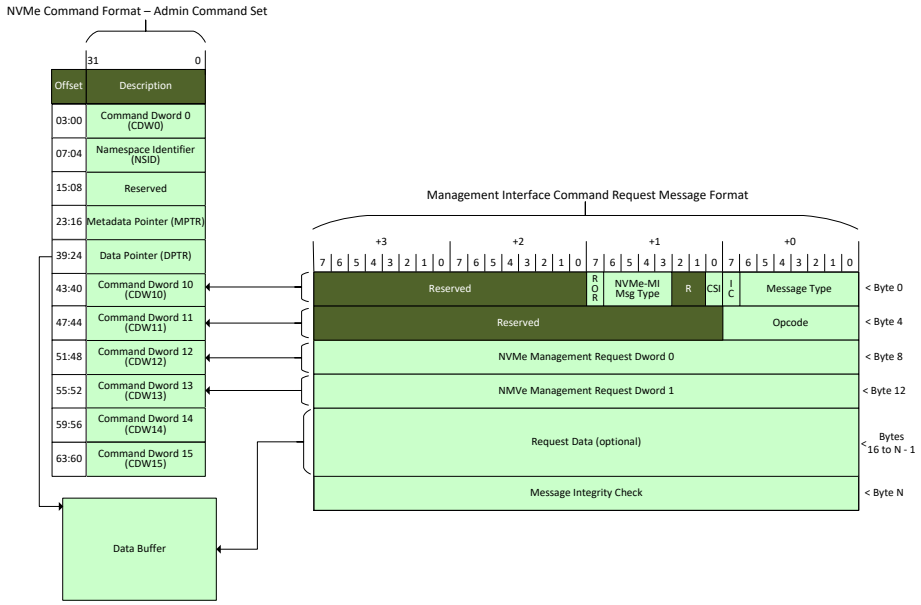


Figure 4.3.1.1_2 NVMe-MI Send Command Request Message to NVMe Admin Command SQE Mapping Table

NVMe-MI Command Request Message		NVMe Admin Command SQE Mapping	
Byte	**Description**	**Byte**	**Description**
NA	These bytes have no equivalent in NVMe-MI.	23:00	Refer to the NVM Express specification.
NA	There is no equivalent of DPTR in NVMe-MI. In NVMe-MI Send, the Request Data is included in the Request Data portion of the Request Message.	39:24	**Data Pointer (DPTR):** This field contains a pointer to the start of the data buffer that contains the Request Data portion of the NVMe-MI Command that is being tunneled. If there is no Request Data for this command then this field is ignored. Refer to the NVM Express Specification for the definition of this field.
03:00	NVMe-MI Message Header (NMH)	43:40	**Command Dword 10 (CDW10):** Dword 0 of the Request Message (NMH) that is being tunneled maps to CDW10 of the SQE. The byte ordering within CDW10 is little endian (i.e. NMH byte 0 maps to CDW10 byte 0, NMH byte 1 maps to CDW10 byte 1, etc.).
04	Opcode (OPC)	47:44	

07:05	Reserved		**Command Dword 11 (CDW11):** Dword 1 of the Request Message (OPC and Reserved bytes 7:5) that is being tunneled maps to CDW11 of the SQE. The byte ordering within CDW11 is little endian (i.e. OPC maps to CDW11 byte 0, the LSB of the Reserved field (NVMe-MI Command Request Message byte 5) maps to CDW11 byte 1, etc.).
11:08	NVMe Management Dword 0 (NMD0)	51:48	**Command Dword 12 (CDW12):** Dword 2 of the Request Message (NMD0) that is being tunneled maps to CDW12 of the SQE. The byte ordering within CDW12 is little endian (i.e. NMD0 byte 0 maps to CDW12 byte 0, NMD0 byte 1 maps to CDW12 byte 1.
15:12	NVMe Management Dword 1 (NMD1)	55:52	**Command Dword 13 (CDW13):** Dword 3 of the Request Message (NMD1) that is being tunneled maps to CDW13 of the SQE. The byte ordering within CDW13 is little endian (i.e. NMD1 byte 0 maps to CDW13 byte 0, NMD1 byte 1 maps to CDW13 byte 1.
NA	This field has no equivalent in NVMe-MI.	59:56	**Command Dword 14 (CDW14):** Reserved.
NA	This field has no equivalent in NVMe-MI.	63:60	**Command Dword 15 (CDW15):** Reserved.
N-1:16	Request Data (REQD)	NA	Request Data is placed by host software into the data buffer pointed to by DPTR. If the Request Data is not Dword granular, then the Request Data shall be padded with the minumum number of bytes of zeroes to make the Request Data Dword granular. The byte ordering within the data buffer pointed to by DPTR is little endian (i.e. REQD byte 0 maps to byte 0 of the data buffer pointed to by DPTR, REQD byte 1 maps to byte 1 of the data buffer pointed to by DPTR, etc.).
N+3:N	Message Integrity Check (MIC)	NA	The Message Integrity Check is not used in the in-band tunneling mechanism.

Figure 4.3.1.1_3 NVMe-MI Send Command Response Message to NVMe Admin Command CQE Mapping Diagram

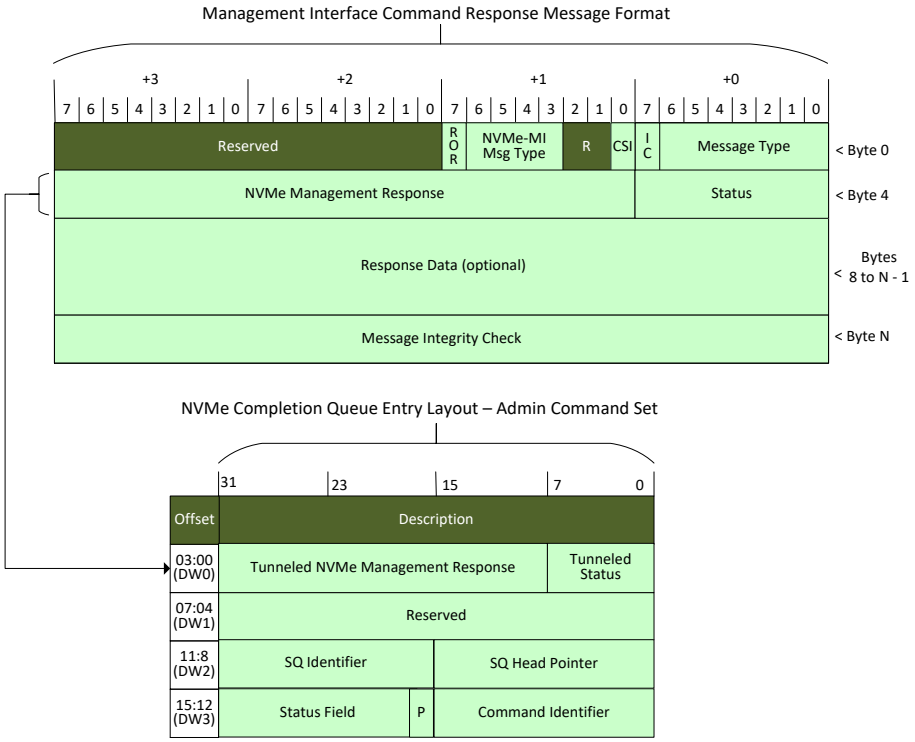


Figure 4.3.1.1_4 NVMe-MI Send Command Response Message to NVMe Admin Command CQE Mapping Table

NVMe-MI Command Response Message		NVMe Admin Command CQE Mapping	
Byte	**Description**	**Byte**	**Description**
00	MCTP Data (MCTPD)	NA	This field has no equivalent in the NVMe Admin Command SQE.
01	NVMe-MI Message Parameters (NMP)	NA	This field has no equivalent in the NVMe Admin Command SQE.
03:02	Reserved	NA	This field has no equivalent in the NVMe Admin Command SQE.
04	Status (STATUS)	03:00	

07:05	NVMe Management Response (NMRESP)		**Command Specific (DW0):** Dword 1 of the Response Message (STATUS and NMRESP) that is being tunneled maps to DW0 of the CQE. The byte ordering within DW0 is little endian (i.e. STATUS maps to DW0 byte 0, the LSB of the NMRESP field (NVMe-MI Command Response Message byte 5) maps to DW0 byte 1, etc.). Refer to Figure 4.3.1.1_5 for additional details on this field.
N-1:8	Response Data (RESPD)	NA	There is no Response Data for NVMe-MI Send.
N+3:N	Message Integrity Check (MIC)	NA	The Message Integrity Check is not used in the in-band tunneling mechanism.
NA	These bytes have no equivalent in NVMe-MI.	15:04	Refer to the NVM Express specification.

The definition of Dword 0 of the completion queue entry is in Figure 4.3.1.1_5.

Figure 4.3.1.1_5: NVMe-MI Send – Completion Queue Entry Dword 0 (NSCQED0)

Bit	Description
31:08	**Tunneled NVMe Management Response (TNMRESP):** This field contains the NVMe Management Response field from the NVMe-MI Command that is being tunneled in-band. If any errors are detected in the NVMe Context as described in section 4.3.1.2 then this field shall be set to 0h.
07:00	**Tunneled Status (TSTAT):** This field contains the Status field from the NVMe-MI Command that is being tunneled in-band. If any errors are detected in the NVMe Context as described in section 4.3.1.2 then this field shall be set to 0h.

4.3.1.2 NVMe-MI Send Command Servicing Model

The NVMe-MI Send command servicing model is illustrated in Figure 4.3.1.2_1 as a series of phases and NVMe/NVMe-MI Contexts. The phases of the NVMe-MI Send command servicing model are further described in this section. The behavior of the portions of the figure in the NVMe Context are specified by the NVM Express specification. The behavior of the portions of the figure in the NVMe-MI Context are specified by this specification. The phases and NVMe/NVMe-MI Contexts are logical constructs that illustrate the NVMe-MI Send command servicing model and do not mandate a particular implementation.

This section describes the NVMe-MI Send command servicing model starting at NVMe Processing as shown in phase 1 of Figure 4.3.1.2_1. In phase 1, CDW0 to CDW9 are checked for errors per the NVM Express specification. If any errors are encountered in CDW0 to CDW9, then the NVMe-MI Send command is completed with an error status code in the Status Field as per the NVM Express specification and the Tunneled Status and Tunneled NVMe Management Response fields shall be set to 0h.

If there are no errors in CDW0 to CDW9, then command servicing enters phase 2 where the portion of the tunneled NVMe-MI Command in CDW10 to CDW15 is checked for errors. Note that if there is no Request Data then CDW10 to CDW15 contain the entire tunneled NVMe-MI Command. If any errors are encountered in the portion of the tunneled NVMe-MI Command in CDW10 to CDW15, then the NVMe-MI Send command is completed with a status code of Successful Completion in the Status Field as defined in the NVM Express specification. The Tunneled Status field contains the error Response Message Status Value for the portion of the tunneled NVMe-MI Command in CDW10 to CDW15 and the Tunneled NVMe Management Response field contains the NVMe Management Response field from the NVMe-MI

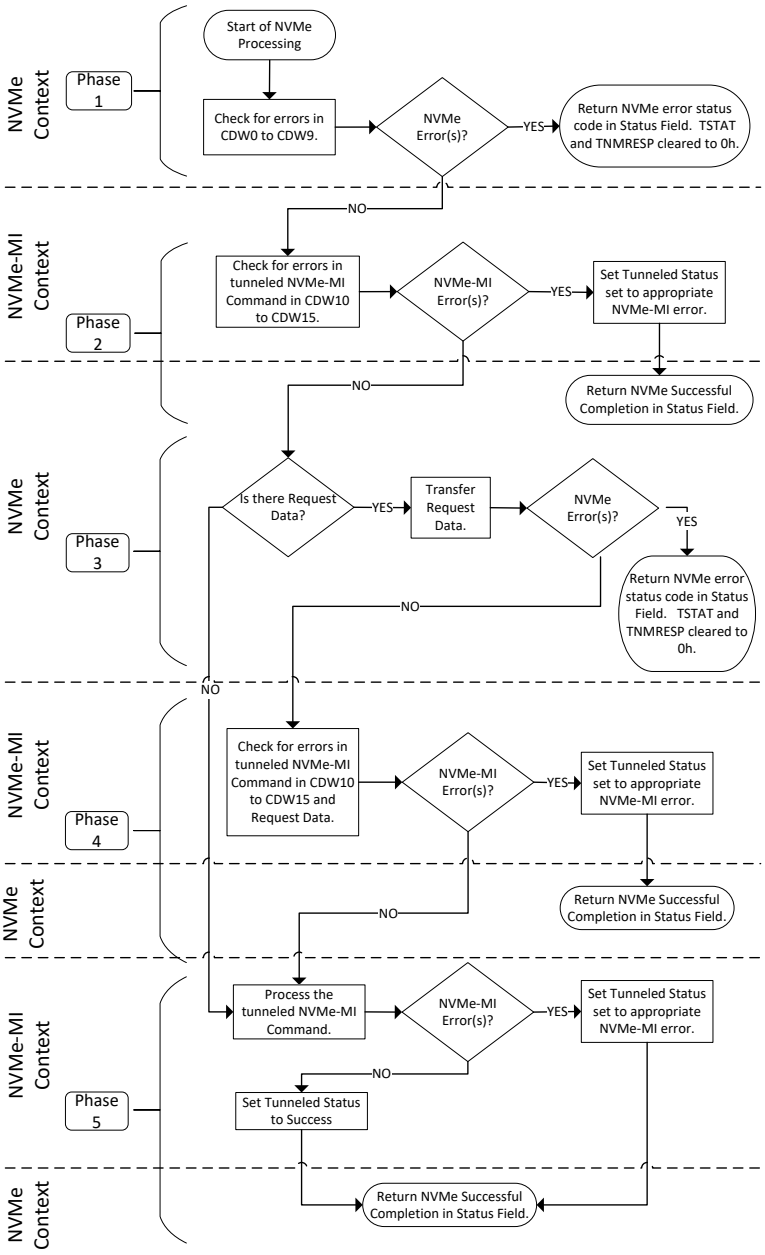
Command that is being tunneled in-band.

If there are no errors in phase 2, then command servicing enters phase 3 where there is a check to determine if there is any Request Data for the tunneled NVMe-MI Command. If there is no Request Data for the tunneled NVMe-MI Command, then command servicing skips to phase 5. If there is Request Data, then the Request Data is transferred from the buffer pointed to by DPTR. If any errors are encountered transferring the Request Data then the command is completed with an error status code in the Status Field as per the NVM Express specification and the Tunneled Status and Tunneled NVMe Management Response fields shall be set to 0h.

If there are no errors transferring the data, then command servicing enters phase 4 where the whole tunneled NVMe-MI Command is constructed from CDW10 to CDW15 and the Request Data that was transferred. If any errors are encountered in the tunneled NVMe-MI Command, then the NVMe-MI Send command is completed with a status code of Successful Completion in the Status Field as defined in the NVM Express specification. The Tunneled Status field contains the appropriate error Response Message Status Value and the Tunneled NVMe Management Response field contains the NVMe Management Response field from the NVMe-MI Command that is being tunneled in-band.

If there are no errors in phase 4, then command servicing enters phase 5 where the tunneled NVMe-MI Command finishes processing. If any errors are encountered processing the tunneled NVMe-MI Command, then the NVMe-MI Send command is completed with a status code of Successful Completion in the Status Field as defined in the NVM Express specification and the Tunneled Status field contains the appropriate error Response Message Status Value. If the tunneled NVMe-MI Command is processed successfully, then the NVMe-MI Send command is completed with a status code of Successful Completion in the Status Field as defined in the NVM Express specification. The Tunneled Status field contains a Response Message Status Value of Success for the tunneled NVMe-MI Command and the Tunneled NVMe Management Response field contains the NVMe Management Response field from the NVMe-MI Command that is being tunneled in-band.

Figure 4.3.1.2_1: NVMe-MI Send Command Servicing Model



4.3.2 NVMe-MI Receive Command

The NVMe-MI Receive command is an NVMe Admin Command as defined by this specification and the NVM Express Specification. It is used to tunnel an NVMe-MI Command in-band from host software to an NVMe Controller that transfers data from an NVMe Controller to a host (similar to a read operation). The data being transferred is in one or more of the following locations: Response Data, NVMe Management Response.

The mapping of how NVMe-MI Commands are tunneled inside of NVMe-MI Receive commands is described in section 4.3.2.1. The NVMe-MI Receive command servicing model is described in section 4.3.2.2.

4.3.2.1 NVMe-MI Receive Command Request Message to NVMe Admin Command SQE Mapping

In order to tunnel NVMe-MI Commands in-band via NVMe-MI Receive, NVMe-MI Request Messages are mapped onto NVMe Submission Queue Entries (SQE) as shown pictorially in Figure 4.3.2.1_1 and in table form in Figure 4.3.2.1_2. NVMe-MI Response Messages are mapped on to NVMe Completion Queue Entries (CQE) as shown pictorially in Figure 4.3.2.1_1 and in table form in Figure 4.3.2.1_3. Refer to the NVM Express specification for details on NVMe Submission Queue Entries and NVMe Completion Queue Entries.

Figure 4.3.2.1_1 NVMe-MI Receive Command Request/Response Message to NVMe Admin Command SQE/CQE Mapping Diagram

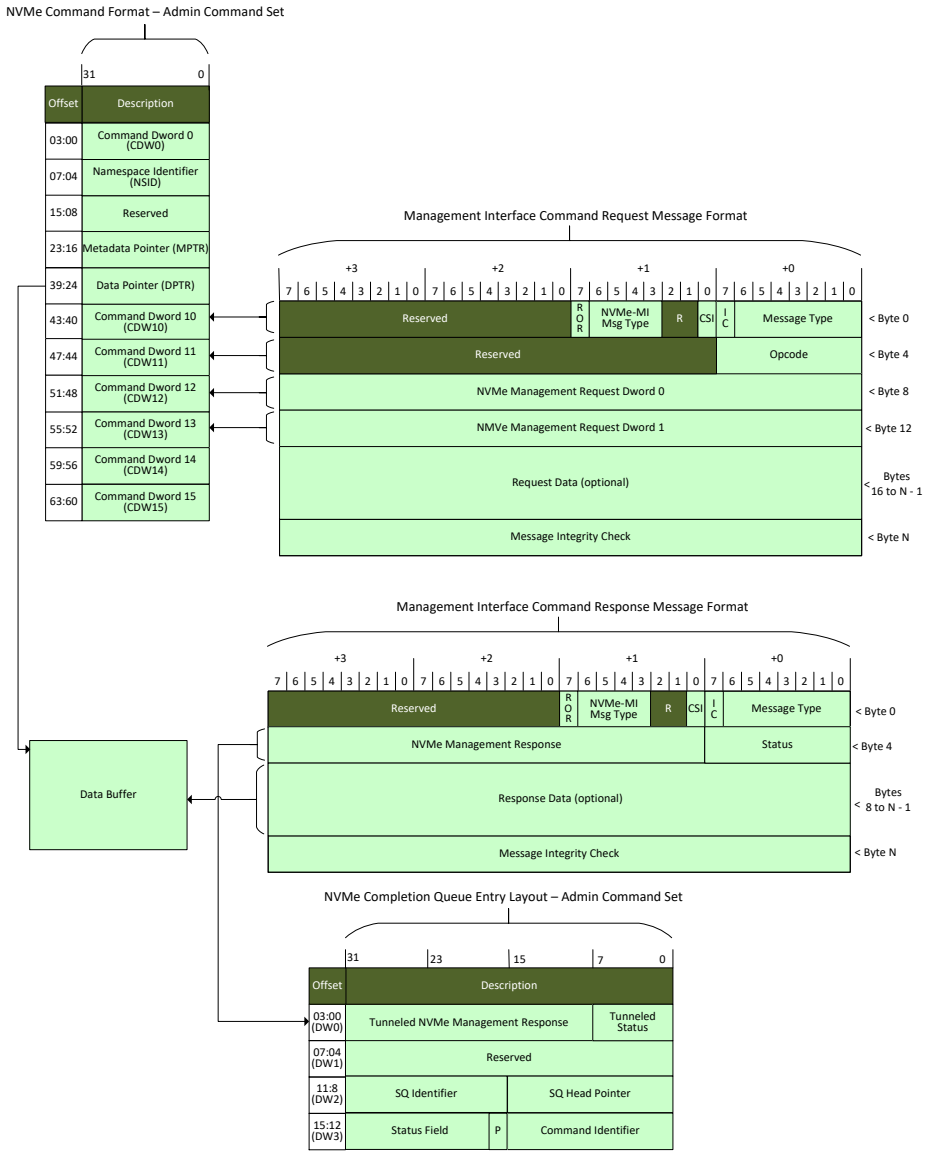


Figure 4.3.2.1_2 NVMe-MI Receive Command Request/Response Message to NVMe Admin Command SQE/CQE Mapping Table

NVMe-MI Command Request Message		NVMe Admin Command SQE Mapping	
Byte	**Description**	**Byte**	**Description**
NA	These bytes have no equivalent in NVMe-MI.	23:00	Refer to the NVM Express specification.
NA	There is no equivalent of DPTR in NVMe-MI. In NVMe-MI Receive, the Response Data is included in the Response Data portion of the Response Message.	39:24	**Data Pointer (DPTR):** This field contains a pointer to the start of the data buffer that contains the Response Data portion of the NVMe-MI Command that is being tunneled. If there is no Resposne Data for this command then this field is ignored. Refer to the NVM Express Specification for the definition of this field.
03:00	NVMe-MI Message Header (NMH)	43:40	**Command Dword 10 (CDW10):** Dword 0 of the Request Message (NMH) that is being tunneled maps to CDW10 of the SQE. The byte ordering within CDW10 is little endian (i.e. NMH byte 0 maps to CDW10 byte 0, NMH byte 1 maps to CDW10 byte 1, etc.).
04	Opcode (OPC)	47:44	**Command Dword 11 (CDW11):** Dword 1 of the Request Message (OPC and Reserved bytes 7:5) that is being tunneled maps to CDW11 of the SQE. The byte ordering within CDW11 is little endian (i.e. OPC maps to CDW11 byte 0, the LSB of the Reserved field (NVMe-MI Command Request Message byte 5) maps to CDW11 byte 1, etc.).
07:05	Reserved		
11:08	NVMe Management Dword 0 (NMD0)	51:48	**Command Dword 12 (CDW12):** Dword 2 of the Request Message (NMD0) that is being tunneled maps to CDW12 of the SQE. The byte ordering within CDW12 is little endian (i.e. NMD0 byte 0 maps to CDW12 byte 0, NMD0 byte 1 maps to CDW12 byte 1.
15:12	NVMe Management Dword 1 (NMD1)	55:52	**Command Dword 13 (CDW13):** Dword 3 of the Request Message (NMD1) that is being tunneled maps to CDW13 of the SQE. The byte ordering within CDW13 is little endian (i.e. NMD1 byte 0 maps to CDW13 byte 0, NMD1 byte 1 maps to CDW13 byte 1.
NA	This field has no equivalent in NVMe-MI.	59:56	**Command Dword 14 (CDW14):** Reserved.
NA	This field has no equivalent in NVMe-MI.	63:60	**Command Dword 15 (CDW15):** Reserved.
N-1:16	Request Data (REQD)	NA	There is no Request Data for NVMe-MI Receive.
N+3:N	Message Integrity Check (MIC)	NA	The Message Integrity Check is not used in the in-band tunneling mechanism.

Figure 4.3.2.1_3 NVMe-MI Receive Command Response Message to NVMe Admin Command CQE Mapping Table

NVMe-MI Command Response Message	NVMe Admin Command CQE

Byte	Description	Byte	Description
00	MCTP Data (MCTPD)	NA	This field has no equivalent in the NVMe Admin Command SQE.
01	NVMe-MI Message Parameters (NMP)	NA	This field has no equivalent in the NVMe Admin Command SQE.
03:02	Reserved	NA	This field has no equivalent in the NVMe Admin Command SQE.
04	Status (STATUS)	03:00	**Command Specific (DW0):** Dword 1 of the Response Message (STATUS and NMRESP) that is being tunneled maps to DW0 of the CQE. The byte ordering within DW0 is little endian (i.e. STATUS maps to DW0 byte 0, the LSB of the NMRESP field (NVMe-MI Command Response Message byte 5) maps to DW0 byte 1, etc.). Refer to Figure 4.3.2.1_4 for additional details on this field.
07:05	NVMe Management Response (NMRESP)		
N-1:8	Response Data (RESPD)	NA	Response Data is placed by the NVMe Controller into the data buffer pointed to by DPTR. If the Response Data size is not Dword granular, then the Response Data shall be padded with the minimum number of bytes of zeroes to make the Response Data Dword granular. The byte ordering within the data buffer pointed to by DPTR is little endian (i.e. RESPD byte 0 maps to byte 0 of the data buffer pointed to by DPTR, RESPD byte 1 maps to byte 1 of the data buffer pointed to by DPTR, etc.).
N+3:N	Message Integrity Check (MIC)	NA	The Message Integrity Check is not used in the in-band tunneling mechanism.
NA	These bytes have no equivalent in NVMe-MI.	15:04	Refer to the NVM Express specification.

The definition of Dword 0 of the completion queue entry is in Figure 4.3.2.1_4.

Figure 4.3.2.1_4: NVMe-MI Receive – Completion Queue Entry Dword 0 (NRCQED0)

Bit	Description
31:08	**Tunneled NVMe Management Response (TNMRESP):** This field contains the NVMe Management Response field from the NVMe-MI Command that is being tunneled in-band. If any errors are detected in the NVMe Context as described in section 4.3.2.2 then this field shall be set to 0h.
07:00	**Tunneled Status (TSTAT):** This field contains the Status field from the NVMe-MI Command that is being tunneled in-band. If any errors are detected in the NVMe Context as described in section 4.3.2.2 then this field shall be set to 0h.

4.3.2.2 NVMe-MI Receive Command Servicing Model

The NVMe-MI Receive command servicing model is illustrated in Figure 4.3.2.2_1 as a series of phases (described in this section) and NVMe/NVMe-MI Contexts. The phases of the NVMe-MI Receive command servicing model are further described in this section. The behavior of the portions of the figure in the NVMe Context are specified by the NVM Express specification. The behavior of the portions of the figure in the NVMe-MI Context are specified by this specification. The phases and NVMe/NVMe-MI Contexts are logical

constructs that illustrate the NVMe-MI Receive command servicing model and do not mandate a particular implementation.

This section describes the NVMe-MI Receive command servicing model starting at NVMe Processing as shown in phase 1 of Figure 4.3.2.2_1. In phase 1, CDW0 to CDW9 are checked for errors per the NVM Express specification. If any errors are encountered in CDW0 to CDW9, then the command is completed with an error status code in the Status Field as per the NVM Express specification and the Tunneled Status and Tunneled NVMe Management Response fields shall be set to 0h.

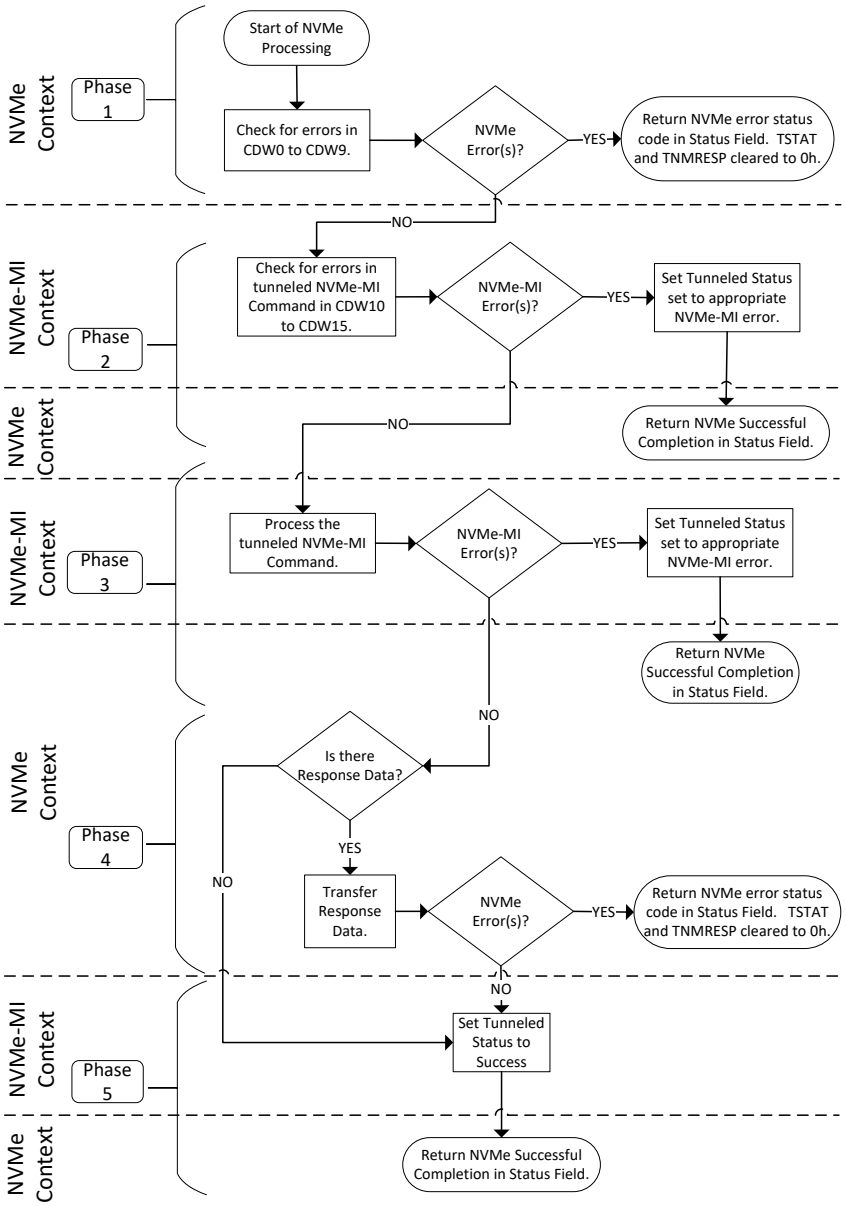
If there are no errors in CDW0 to CDW9, then command servicing enters phase 2 where the tunneled NVMe-MI Command in CDW10 to CDW15 is checked for errors. If any errors are encountered in the tunneled NVMe-MI Command in CDW10 to CDW15, then the NVMe-MI Receive command is completed with a status code of Successful Completion in the Status Field as defined in the NVM Express specification. The Tunneled Status field contains the appropriate error Response Message Status Value and the Tunneled NVMe Management Response field contains the NVMe Management Response field from the NVMe-MI Command that is being tunneled in-band.

If there are no errors in phase 2, then command servicing enters phase 3 where the tunneled NVMe-MI Command finishes processing. If any errors are encountered processing the tunneled NVMe-MI Command, then the NVMe-MI Receive command is completed with a status code of Successful Completion in the Status Field as defined in the NVM Express specification. The Tunneled Status field contains the appropriate error Response Message Status Value and the Tunneled NVMe Management Response field contains the NVMe Management Response field from the NVMe-MI Command that is being tunneled in-band.

If there are no errors in phase 3, then command servicing enters phase 4 where there is a check to determine if there is any Response Data for the tunneled NVMe-MI Command. If there is no Response Data for the tunneled NVMe-MI Command, then command servicing skips to phase 5. If there is Response Data, then the Response Data is transferred to the buffer pointed to by DPTR. If any errors are encountered transferring the Response Data then the command is completed with an error status code in the Status Field as per the NVM Express specification and the Tunneled Status and Tunneled NVMe Management Response fields shall be set to 0h.

If there are no errors in phase 4, then command servicing enters phase 5 where the NVMe-MI Receive command is completed with a status code of Successful Completion in the Status Field as defined in the NVM Express specification. The Tunneled Status field contains a Response Message Status Value of Success for the tunneled NVMe-MI Command and the Tunneled NVMe Management Response field contains the NVMe Management Response field from the NVMe-MI Command that is being tunneled in-band.

Figure 4.3.2.2_1: NVMe-MI Receive Command Servicing Model



Modify Section 5 (Management Interface Command Set) as shown below:

The Management Interface Command Set defines the Command Messages that may be submitted by a Requester~~Management Controller~~ when the NMIMT value is set to NVMe-MI Command. The Management Interface Command Set is applicable to both the out-of-band mechanism and the in-band tunneling mechanism.

The ~~MCTP~~NVMe-MI Message structure with all fields that are common to all ~~MCTP~~NVMe-MI Messages are defined ~~for commands~~ in section 3.2. The Response Message structure for the Management Interface Command Set is defined in section 4.2. The Message Body for the Management Interface Command~~s~~ Set is shown in Figure 34. Command specific fields for the Management Interface ~~c~~Command ~~s~~Set are defined in this section.

Commented [AB248]: Section 3.1 after sections are moved.

Commented [AB249]: Section 4.1.2 after sections are moved.

Figure 33: ~~Management Interface~~ NVMe-MI Command Request Message Format

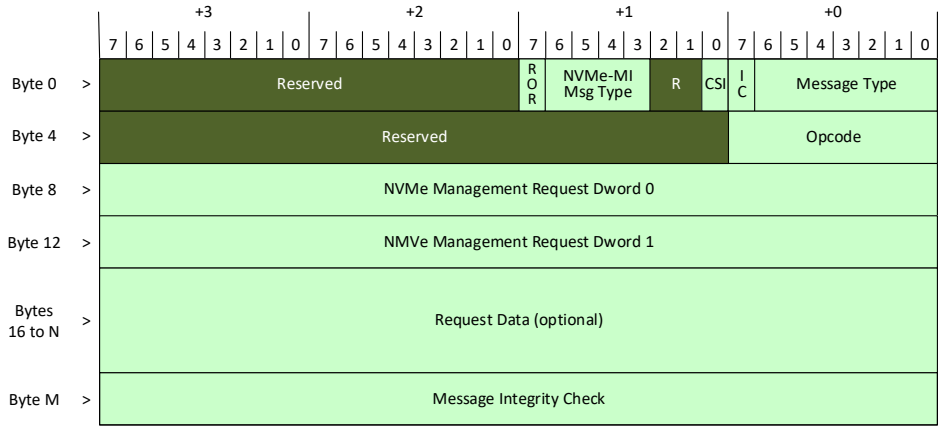


Figure 34: ~~Management Interface~~ NVMe-MI Command Request Message Description (NCREQ)

Byte	Description
03:00	**NVMe-MI Message Header (NMH)**: Refer to 3.2.
04	**Opcode (OPC):** This field specifies the opcode of the NVMe-MI ~~Management Interface~~ ~~c~~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 (REQD):** (Optional)
M+3:M	**Message Integrity Check (MIC):** Refer to 3.2.

Commented [AB250]: Section 3.1 after sections are moved.

Commented [AB251]: Section 3.1 after sections are moved.

The Request Data field is an optional field included in some NVMe-MI ~~Management Interface c~~Commands. If the size of the Request Data does not match the specified Data Length of the Command Message, then the ~~Management Endpoint~~Responder responds with a ~~g~~Generic ~~e~~Error ~~r~~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	O	Reset
08h – BFh	-	Reserved
C0h – FFh	O	Vendor specific
NOTES: 1. O/M definition: O = Optional, M = Mandatory.		

Figure 36: ~~Management Interface~~ NVMe-MI Command Response Message Format

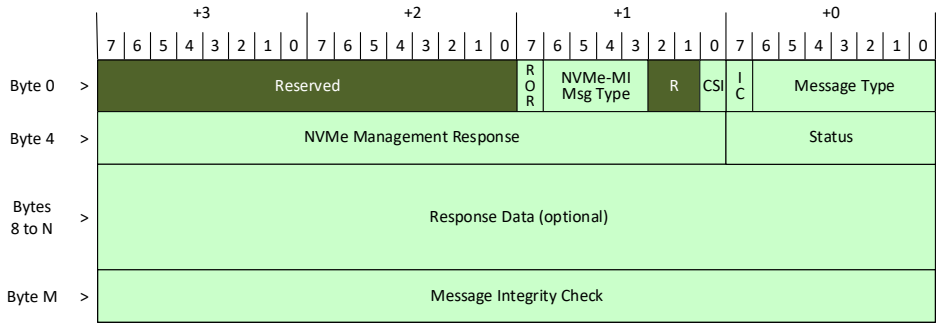


Figure 37: ~~Management Interface~~ NVMe-MI Command Response Message Description (NCRESP)

Byte	Description
03:00	**NVMe-MI Message Header (NMH)**: Refer to 3.2
04	**Status (STATUS):** This field indicates the status of the NVMe-MI ~~c~~Command. Refer to 4.2.
07:05	**NVMe Management Response (NMRESP):** This field is command specific.
N:8	**Response Data (RESPD):** (optional)
M+3:M	**Message Integrity Check (MIC):** Refer to 3.2.

Commented [AB252]: Section 3.1 after sections are moved.

Commented [AB253]: Section 4.1.2 after sections are moved.

Commented [AB254]: Section 3.1 after sections are moved.

Modify Figure 40 (NVMe Management Interface Configuration Identifiers) as shown below:

~~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:~~ ~~1. O/M definition: O = Optional, M = Mandatory.~~		

Configuration Identifier	O/M[1] Out-of-Band Mechanism	O/M[1] In-Band Tunneling Mechanism	Description
00h	-	-	Reserved
01h	M	P	SMBus/I2C Frequency
02h	M	M	Health Status Change
03h	M	P	MCTP Transmission Unit Size
04h - BFh	-	-	Reserved
C0h - FFh	O	O	Vendor Specific
NOTES: 1. O/M/P definition: O = Optional, M = Mandatory, P = Prohibited from being supported.			

Modify Section 5.1.2 (Health Status Change (Configuration Identifier 02h)) as shown below:

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~~Requester 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~~Responder 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.

Modify a portion of Section 5.2 (Configuration Set) as shown below:

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

Modify a portion of Figure 64 (PCIe Port Specific data) as shown below:

11	**PCIe Maximum Link Width:** 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~~Requester may compare this value with the PCIe Negotiated Link Width to determine if there has been a PCIe link training issue.

Value	Definition
0	Reserved
1	PCIe x1
2	PCIe x2
3	Reserved
4	PCIe x4
5-7	Reserved
8	PCIe x8
9-11	Reserved
12	PCIe x12
13-15	Reserved
16	PCIe x16
17-31	Reserved
32	PCIe x32
33-255	Reserved

Modify a portion of Section 5.5 (Read NVMe-MI Data Structure) as shown below:

The Optionally Supported Command List data structure contains a list of optional commands that a ~~Management Endpoint~~Responder 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).

Modify a portion of Section 5.7 (VPD Read) as shown below:

A VPD Read command with length 0 and no data is valid. The ~~Management Endpoint~~Responder 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~~Responder does not return the VPD contents and responds with an Invalid Parameter error status response.

Modify a portion of Section 5.8 (VPD Write) as shown below:

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~~Responder responds with a ~~g~~Generic ~~e~~Error ~~r~~Response and VPD Writes Exceeded status.

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

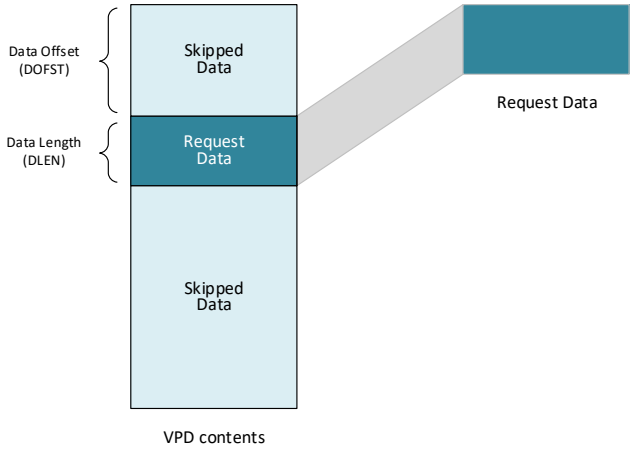
Figure 73: VPD Write – NVMe Management Dword 0

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 74: 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 3: VPD Write Request Data



The ~~Management Controller~~Requester should not read the contents of the VPD while this command is ~~processing~~servicing. 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~~Responder does not write to the VPD and responds with an Invalid Parameter error status response.

Modify a portion of Section 6 (NVM Express Admin Command Set) as shown below:

Future revisions of this specification may add additional commands to Figure 76. The NVM Express Admin Command Set is only applicable in the out-of-band mechanism and is prohibited in the in-band tunneling mechanism.

Modify a portion of Section 7 (PCIe Command Set (optional)) as shown below:

The NMIMT field in the message header for PCIe Command Messages and Response Messages is set to

4h (PCIe Command). The PCIe Command Set is only applicable in the out-of-band mechanism and is prohibited in the in-band tunneling mechanism.

Modify Figure 86 (PCIe Command Response Description) as shown below:

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

Commented [AB255]: Section 3.1 after sections are moved.

Commented [AB256]: Section 4.1.2 after sections are moved.

Commented [AB257]: Section 3.1 after sections are moved.

Modify a portion of Section 9.1 (Operational Times) as shown below:

9.1 Out-of-Band Operational Times

In the out-of-band mechanism, ~~T~~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.

Modify a portion of Section 9.2 (Vital Product Data) as shown below:

Each NVM Subsystem with one or more ~~Management Endpoints~~Responders shall have a FRU Information Device which is compliant with the IPMI Platform Management FRU Information Storage Definition.

Modify Section 9.3 (Reset) as shown below:

9.3 Out-of-Band Reset

This section describes NVMe-MI architected resets in the out-of-band mechanism.

Modify a portion of Section 9.3.2 (Controller Level Reset) as shown below:

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 Messages~~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.

Modify a portion of Section 9.4 (Security) as shown below:

9.3 Out-of-Band Security

In the out-of-band mechanism, ~~T~~the Management Endpoint may respond with a Response Message Status value of Access Denied in an ~~e~~Error ~~r~~Response.