LEGAL NOTICE:

This erratum to the NVM Express revision 1.2 specification 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 erratum to the NVM Express revision 1.2 specification 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 - 2016 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 Workgroup
c/o Virtual, Inc.
401 Edgewater Place, Suite 600
Wakefield, MA 01880
info@nvmexpress.org

NVM Express™ Technical Errata

Errata ID	**009**
Revision Date	**4/25/2016**
Affected Spec Ver.	**NVM Express™ 1.2b**
Corrected Spec Ver.	

Errata Author(s)

Name	Company
Judy Brock	Samsung
Nadesan Narenthiran	WDC
Fred Knight	NetApp
Ken Okin	KOC
David Black	EMC
Austin Bolen	Dell
Robert Qiuxin	Huawei

Errata Overview

The erratum clarifies the layout and format of several identifiers including OUI, EUI64, and NGUID.

The specification uses the term Queue entry to indicate either a memory location in the Queue or the contents of that location. The erratum redefines the memory location as the Queue slot and leaves the contents of the slot as the Queue entry. The changes consistently use the words submission and consumption to describe the handoff of commands and completions between the host and the controller. The changes also create consistent capitalization and definition of the Phase Tag bit.

The erratum clarifies that the Format In Progress indicator and error status are associated with a Format NVM command that is in progress for the namespace.

The erratum contains various minor grammatical edits and clarifications.

Revision History

Revision Date	Change Description
3/30/2016	Initial draft
4/7/2016	Updates to the identifier modifications/examples. Added Queue entry corrections.
4/13/2016	Added clarifications that Format in Progress is associated with a Format NVM command in progress for the namespace. Minor editorial / clarification updates.
4/20/2016	Minor additions and edits to text on SQE and CQE lifetimes.
4/25/2016	Red-line accept.

Description of Specification Changes

Modify the beginning of section 5.11 (Identify command) as shown below:

The Identify command returns a data buffer that describes information about the NVM subsystem, the controller or the namespace(s). The data structure is 4096 bytes in size.

The data structure returned, defined in **Error! Reference source not found.**, is based on the Controller or Namespace Structure (CNS) field. If there are fewer namespace identifiers or controller identifiers to return for a Namespace List or Controller List, respectively, then the unused portion of the list is zero filled. Controllers that support Namespace Management shall support CNS values of 10h – 13h.

The Identify Controller data structure and Identify Namespace data structure include several identifiers. The format and layout of these identifiers is described in section 7.10.

Modify a portion of Figure 92 (Identify Namespace) as shown below:

119:104	O	**Namespace Globally Unique Identifier (NGUID):** This field contains a 128-bit value that is globally unique and assigned to the namespace when the namespace is created. This field remains fixed throughout the life of the namespace and is preserved across namespace and controller operations (e.g., controller reset, namespace format, etc.). This field uses the EUI-64 based 16-byte designator format. Bytes 114:112 contain the 24-bit Organizationally Unique Identifier (OUI) value assigned by the IEEE Registration Authority. Bytes 119:115 contain an extension identifer assigned by the corresponding organization. Bytes 111:104 contain the vendor specific extension identifier assigned by the corresponding organization. See the IEEE EUI-64 guidelines for more information. This field is big endian (refer to section 7.10). The controller shall specify a globally unique namespace identifier in this field or the EUI64 field when the namespace is created.

127:120	O	**IEEE Extended Unique Identifier (EUI64):** This field contains a 64-bit IEEE Extended Unique Identifier (EUI-64) that is globally unique and assigned to the namespace when the namespace is created. This field remains fixed throughout the life of the namespace and is preserved across namespace and controller operations (e.g., controller reset, namespace format, etc.). The EUI-64 is a concatenation of a 24-bit or 36-bit Organizationally Unique Identifier (OUI or OUI-36) value assigned by the IEEE Registration Authority and an extension identifier assigned by the corresponding organization. See the IEEE EUI-64 guidelines for more information. This field is big endian (refer to section 7.10). The controller shall specify a globally unique namespace identifier in this field or the NGUID field when the namespace is created. If the controller is not able to allocate a globally unique 64-bit identifier then this field shall be cleared to 0h. Refer to section **Error! Reference source not found.**.

Add section 7.10 as shown below:

7.10 Identifier Format and Layout (Informative)

This section provides guidance for proper implementation of various identifiers defined in the Identify Controller and Identify Namespace data structures.

7.10.1 PCI Vendor ID (VID) and PCI Subsystem Vendor ID (SSVID)

The PCI Vendor ID (VID, bytes 01:00) and PCI Subsystem Vendor ID (SSVID, bytes 03:02) are defined in the Identify Controller data structure. The values are assigned by the PCI SIG. Each identifier is a 16-bit number in little endian format.

Example:

- VID = ABCDh
- SSVID = 1234h

Byte	00	01	02	03
Value	CDh	ABh	34h	12h

7.10.2 Serial Number (SN) and Model Number (MN)

The Serial Number (SN, bytes 23:04) and Model Number (MN, bytes 63:24) are defined in the Identify Controller data structure. The values are ASCII strings assigned by the vendor. Each identifier is in big endian format.

Example (Value shown as ASCII characters):

- SN = “SN1”
- MN = “M2”

Byte	04	05	06	23 - 07	24	25	63 - 26
Value	53h (‘S’)	4Eh (‘N’)	31h (‘1’)	20h (‘ ‘)	4Dh (‘M’)	32h (‘2’)	20h (‘ ‘)

7.10.3 IEEE OUI Identifier (IEEE)

The IEEE OUI Identifier (OUI, bytes 75:73) is defined in the Identify Controller data structure. The value is assigned by the IEEE Registration Authority. The identifier is in little endian format.

Example:

- OUI = ABCDEFh

Byte	73	74	75
Value	EFh	CDh	ABh

7.10.4 IEEE Extended Unique Identifier (EUI64)

The IEEE Extended Unique Identifier (EUI64, bytes 127:120) is defined in the Identify Namespace data structure. A tutorial is available at https://standards.ieee.org/develop/regauth/tut/eui64.pdf. IEEE defines three formats that may be used in this field: MA-L, MA-M, and MA-S. The examples in this section uses the MA-L format.

The MA-L format is defined as a string of eight octets:

EUI[0	EUI[1]	EUI[2]	EUI[3]	EUI[4]	EUI[5]	EUI[6]	EUI[7]
OUI			Extension Identifier				

EUI64 is defined in big endian format. The OUI field differs from the OUI Identifier which is in little endian format as described in section 7.10.3.

Example:

- OUI Identifier = ABCDEFh
- Extension Identifier = 0123456789h

Byte	120	121	122	123	124	125
Value	ABh	CDh	EFh	01h	23h	45h
Field	OUI			Extension Identifier		

Byte	126	127
Value	67h	89h
Field	Ext ID (cont)	

The MA-L format is similar to the World Wide Name (WWN) format defined as IEEE Registered designator (NAA = 5) as shown below.

Byte	0	1	2	3	4	5	6	7
EUI64	OUI			Extension Identifier				
WWN (NAA = 5)	5h	OUI		Vendor Specific Identifier				

7.10.5 Namespace Globally Unique Identifier (NGUID)

The Namespace Globally Unique Identifier (NGUID, bytes 119:104) is defined in the Identify Namespace data structure. The NGUID is composed of an IEEE OUI, an extension identifier, and a vendor specific extension identifier. The extension identifier and vendor specific extension identifier are both assigned by the vendor and may be considered as a single field. NGUID is defined in big endian format. The OUI field differs from the OUI Identifier which is in little endian format as described in section 7.10.3.

Example:

- OUI Identifier = ABCDEFh
- Extension Identifier = 0123456789h
- Vendor Specific Extension Identifier = FEDCBA9876543210h

Byte	104	105	106	107	108	109
Value	FEh	DCh	BAh	98h	76h	54h
Field	Vendor Specific Extension Identifier					

Byte	110	111	112	113	114	115
Value	32h	10h	ABh	CDh	EFh	01h
Field	VSP Ex ID (cont)		OUI			Ex ID

Byte	116	117	118	119
Value	23h	45h	67h	89h
Field	Extension Identifier (cont)			

The NGUID format is similar to the World Wide Name (WWN) format as IEEE Registered Extended designator (NAA = 6) as shown below.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NGUID	Vendor Specific Extension Identifier								OUI			Extension Identifier				
WWN (NAA = 6)	6h	OUI			Vendor Specific Identifier				Vendor Specific Identifier Extension							

Modify section 1.6.4 (candidate command) as shown below:

A candidate command is a submitted command which has been transferred into the controller and the controller deems ready for processing.

Modify section 1.6.6 (command submission) as shown below:

A command is submitted when a Submission Queue Tail Doorbell write has completed that moves the Submission Queue Tail Pointer value past the ~~corresponding~~ Submission Queue ~~entry for~~ slot in which the ~~associated~~ command was placed.

Modify the last paragraph of section 1.4 (Theory of Operation) as shown below:

A Completion Queue (CQ) is a circular buffer with a fixed slot size used to post status for completed commands. A completed command is uniquely identified by a combination of the associated SQ identifier and command identifier that is assigned by host software. Multiple Submission Queues may be associated with a single Completion Queue. This feature may be used where a single worker thread processes all command completions via one Completion Queue even when those commands originated from multiple Submission Queues. The CQ Head pointer is updated by host software after it has processed completion queue entries indicating the last free CQ slot ~~entry~~. A Phase Tag (P) bit is defined in the completion queue entry to indicate whether an entry has been newly posted without consulting a register. This enables host software to determine whether the new entry was posted as part of the previous or current round of completion notifications. Specifically, each round through the Completion Queue entries, the controller inverts the Phase Tag bit.

Modify section 4.1 (Submission Queue & Completion Queue Definition) as shown below:

~~The Head and Tail entry pointers correspond to the Completion Queue Head Doorbells and the Submission Queue Tail Doorbells defined in section 3.1.14 and 3.1.13. The doorbell registers are updated by host software.~~

The submitter of entries to a queue uses the current Tail entry pointer to identify the next open queue ~~entry space~~ slot. The submitter increments the Tail entry pointer after ~~submitting~~ placing the new entry to the open queue ~~entry space~~ slot. If the Tail entry pointer increment exceeds the queue size, the Tail entry shall roll to zero. The submitter may continue to ~~submit~~ place entries ~~to the queue~~ in free queue slots as long as the Full queue condition is not met (refer to section 4.1.2).

Note: The submitter shall take queue wrap conditions into account.

The consumer of entries on a queue uses the current Head entry pointer to identify the slot containing the next entry to be ~~pulled off the queue~~ consumed. The consumer increments the Head entry pointer after ~~retrieving~~ consuming the next entry from the queue. If the Head entry pointer increment exceeds the queue size, the Head entry pointer shall roll to zero. The consumer may continue to ~~remove~~ consume entries from the queue as long as the Empty queue condition is not met (refer to section 4.1.1).

Note: The consumer shall take queue wrap conditions into account.

Creation and deletion of Submission Queue and associated Completion Queues need to be ordered correctly by host software. Host software shall create the Completion Queue before creating any associated Submission Queue. Submission Queues may be created at any time after the associated Completion Queue is created. Host software shall delete all associated Submission Queues prior to deleting a Completion Queue. To abort all commands submitted to the Submission Queue host software should issue a Delete I/O Submission Queue Command for that queue (refer to section 7.4.3).

Host software writes the Submission Queue Tail Doorbell (refer to section 3.1.13) and the Completion Queue Head Doorbell (refer to section 3.1.14) to communicate new values of the corresponding entry pointers to the controller. If host software writes an invalid value to the Submission Queue Tail Doorbell or Completion Queue Head Doorbell register and an Asynchronous Event Request command is outstanding, then an asynchronous event is posted to the Admin Completion Queue with a status code of Invalid Doorbell Write Value. The associated queue should be deleted and recreated by host software. For a Submission Queue that experiences this error, the controller may complete previously ~~fetched~~ consumed commands; no additional commands are ~~fetched~~ consumed. This condition may be caused by host software attempting to add an entry to a full Submission Queue or remove an entry from an empty Completion Queue.

Host software checks Completion Queue entry Phase Tag (P) bits in memory to determine whether new Completion Queue entries have been posted. The Completion Queue Tail pointer is only used internally by the controller and is not visible to the host. The controller uses the SQ Head Pointer (SQHD) field in Completion Queue entries to communicate new values of the Submission Queue Head Pointer to the host. A new SQHD value indicates that Submission Queue entries have been consumed, but does not indicate either execution or completion of any command. Refer to section 4.6.

~~The behavior if a command is changed between submission and consumption by the controller is undefined. The command has been consumed when a completion entry is posted that moves the Submission Queue Head Pointer past the Submission Queue entry that contains this command.~~

A Submission Queue entry is submitted to the controller when the host writes the associated Submission Queue Tail Doorbell with a new value that indicates that the Submission Queue Tail Pointer has moved to or past the slot in which that Submission Queue entry was placed. A Submission Queue Tail Doorbell write may indicate that one or more Submission Queue entries have been submitted.

A Submission Queue entry has been consumed by the controller when a Completion Queue entry is posted that indicates that the Submission Queue Head Pointer has moved past the slot in which that Submission Queue entry was placed. A Completion Queue entry may indicate that one or more Submission Queue entries have been consumed.

A Completion Queue entry is posted to the Completion Queue when the controller write of that Completion Queue entry to the next free Completion Queue slot inverts the Phase Tag (P) bit from its previous value in memory. The controller may generate an interrupt to the host to indicate that one or more Completion Queue entries have been posted.

A Completion Queue entry has been consumed by the host when the host writes the associated Completion Queue Head Doorbell with a new value that indicates that the Completion Queue Head Pointer has moved past the slot in which that Completion Queue entry was placed. A Completion Queue Head Doorbell write may indicate that one or more Completion Queue entries have been consumed.

Once a Submission Queue or Completion Queue entry has been consumed, the slot in which it was placed is free and available for reuse. Altering a Submission Queue entry after that entry has been submitted but before that entry has been consumed results in undefined behavior. Altering a Completion Queue entry after that entry has been posted but before that entry has been consumed results in undefined behavior.

If there are no free ~~completion queue entries~~ slots in a Completion Queue, then the controller shall not post status to that Completion Queue until ~~completion queue entries~~ slots become available. In this case, the controller may stop processing additional Submission Queue entries associated with the affected Completion Queue until ~~completion queue entries~~ slots become available. The controller shall continue processing for other queues.

Modify section 4.1.3 (Queue Size) as shown below:

The Queue Size is indicated in a 16-bit 0's based field that indicates the number of ~~entries~~ slots in the queue. The minimum size for a queue is two ~~entries~~ slots. The maximum size for either an I/O Submission Queue or an I/O Completion Queue is defined as 64K ~~entries~~ slots, limited by the maximum queue size supported by the controller that is reported in the CAP.MQES field. The maximum size for the Admin Submission and Admin Completion Queue is defined as 4K ~~entries~~ slots. One ~~entry~~ slot in each queue is not available for use due to Head and Tail entry pointer definition.

Modify Figure 27 (Completion Queue Entry) as shown below:

Figure 27: Completion Queue Entry: DW 3

Bit	Description
31:17	**Status Field (SF):** Indicates status for the command that is being completed. Refer to section 4.6.1.
16	**Phase Tag (P):** Identifies whether a Completion Queue entry is new. The Phase Tag values for all Completion Queue entries shall be initialized to '0' by host software prior to setting CC.EN to '1'. When the controller places an entry in the Completion Queue, it shall invert the Phase Tag ~~phase tag~~ to enable host software to discriminate a new entry. Specifically, for the first set of completion queue entries after CC.EN is set to '1' all Phase Tags are set to '1' when they are posted. For the second set of completion queue entries, when the controller has wrapped around to the top of the Completion Queue, all Phase Tags are cleared to '0' when they are posted. The value of the Phase Tag is inverted each pass through the Completion Queue.
15:00	**Command Identifier (CID):** Indicates the identifier of the command that is being completed. This identifier is assigned by host software when the command is submitted to the Submission Queue. The combination of the SQ Identifier and Command Identifier uniquely identifies the command that is being completed. The maximum number of requests outstanding at one time is 64K.

Modify section 4.10 (Fused Operations) as shown below:

Fused operations enable a more complex command by "fusing" together two simpler commands. This feature is optional; support for this feature is indicated in the Identify Controller data structure in Figure 90. In a fused operation, the requirements are:

- The commands shall be executed in sequence as an atomic unit. The controller shall behave as if no other operations have been executed between these two commands.
- The operation ends at the point an error is encountered in either command. If the first command in the sequence failed, then the second command shall be aborted. If the second command in the sequence failed, then the completion status of the first command is sequence specific.
- The LBA range, if used, shall be the same for the two commands. If the LBA ranges do not match, the commands should be aborted with status of Invalid Field in Command.
- The commands shall be inserted next to each other in the same Submission Queue. If the first command is in the last ~~entry~~ slot in the Submission Queue, then the second command shall be the first ~~entry~~ slot in the Submission Queue as part of wrapping around. The Submission Queue Tail doorbell pointer update shall indicate both commands as part of one doorbell update.
- If the host desires to abort the fused operation, the host shall submit an Abort command separately for each of the commands.
- A completion queue entry is posted by the controller for each of the commands.

Whether a command is part of a fused operation is indicated in the Fused Operation field of Command Dword 0 in Figure 10. The Fused Operation field also indicates whether this is the first or second command in the operation.

Modify the first four paragraphs of section 4.11 (Command Arbitration) as shown below:

A command is submitted to the controller when a Submission Queue Tail Doorbell write by the host moves the Submission Queue Tail Pointer past the slot containing the corresponding Submission Queue entry. The controller transfers submitted commands into the controller ~~to the controller's local memory~~ for subsequent processing using a vendor specific algorithm.

A command is being processed when the controller and/or namespace state is being accessed or modified by the command (e.g., a Feature setting is being accessed or modified or a logical block is being accessed or modified).

A command is completed when a Completion Queue entry for the command has been posted to the corresponding Completion Queue. Upon completion, all controller state and/or namespace state modifications made by that command are globally visible to all subsequently submitted commands.

A candidate command is a submitted command which has been transferred into the controller that the controller deems ready for processing. The controller selects command(s) for processing from the pool of submitted commands for each Submission Queue. The commands that comprise a fused operation shall be processed together and in order by the controller. The controller may select candidate commands for processing in any order. The order in which commands are selected for processing does not imply the order in which commands are completed.

Modify a portion of section 7.2.1 (Command Processing) as shown below:

This section describes command submission and completion processing. Figure 209 shows the steps that are followed to ~~issue~~ submit and complete a command. The steps are:

1. The host ~~creates a~~ places one or more commands for execution ~~within the appropriate~~ in the next free Submission Queue slot(s) in memory.
2. The host updates the Submission Queue Tail Doorbell register with the new value of the Submission Queue Tail entry pointer. This indicates to the controller that a new command(s) is submitted for processing.

3. The controller transfers ~~fetches~~ the command(s) from in the Submission Queue slot(s) into the controller ~~from memory~~ for future execution. Arbitration is the method used to determine the Submission Queue from which the controller starts processing the next candidate command(s), refer to section 4.11.
4. The controller then proceeds with execution of the next command(s). Commands may complete out of order (the order submitted or started execution).
5. After ~~the~~ a command has completed execution, the controller places ~~writes~~ a completion queue entry ~~to~~ in the next free slot in the associated Completion Queue. As part of the completion queue entry, the controller indicates the most recent ~~SQ~~ Submission Queue entry that has been ~~fetched~~ consumed by advancing the Submission Queue Head pointer in the completion entry. Each new completion queue entry has a Phase Tag inverted from the previous entry to indicate to the host that this completion queue entry is a new entry.
6. The controller optionally generates an interrupt to the host to indicate that there is a new completion queue entry to consume and process. In the figure, this is shown as an MSI-X interrupt, however, it could also be a pin-based or MSI interrupt. Note that based on interrupt coalescing settings, an interrupt may or may not be ~~indicated~~ generated for ~~the command~~ each new completion queue entry.
7. The host consumes and then processes the new completion queue ~~entry~~ entries in the Completion Queue. This includes taking any actions based on error conditions indicated. The host continues consuming and processing completion queue entries until it encounters a previously consumed entry with a Phase Tag inverted from the value of the current completion queue entries.
8. The host writes the Completion Queue Head Doorbell register to indicate that the completion queue entry has been ~~processed~~ consumed. The host may ~~process~~ consume many entries before updating the associated ~~CQHDBL~~ Completion Queue Head Doorbell register.

Modify a portion of section 7.2.2 (Basic Steps when Building a Command) as shown below:

2. Host software ~~shall~~ writes the corresponding Submission Queue doorbell register (SQxTDBL) to submit one or more commands for processing.

The write to the Submission Queue doorbell register triggers the controller to ~~fetch and process the command~~ consume one or more new commands contained in the Submission Queue entry. The controller indicates the most recent SQ entry that has been ~~fetched~~ consumed as part of reporting completions. Host software may use this information to determine when SQ ~~locations~~ slots may be re-used for new commands.

Modify a portion of section 7.2.4 (Command Related Resource Retirement) as shown below:

As part of reporting completions, the controller indicates the most recent Submission Queue entry that has been ~~fetched~~ consumed. ~~Any~~ Submission Queue ~~entries that are indicated as being fetched~~ slots containing consumed Submission Queue entries are free and may be re-used by host software to submit new commands.

Modify a portion of Figure 90 as shown below:

3071:3040	O	**Power State 31 Descriptor (PSD31):** This field indicates the characteristics of power state 31. The format of this field is defined in **Error! Reference source not found.**.
Vendor Specific		
4095:3072	O	~~**Vendor Specific (VS):** This range of bytes is allocated for vendor specific usage.~~ Vendor Specific

Modify a portion of Figure 92 as shown below:

383:192		Reserved
4095:384	O	~~**Vendor Specific (VS):** This range of bytes is allocated for vendor specific usage.~~ Vendor Specific

Modify Figure 192 as shown below:

Figure 197: Write – Command Dword 13

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31:08</td><td>Reserved</td></tr>
<tr><td>07:00</td><td>Dataset Management (DSM): This field indicates attributes for the dataset that the LBA(s) being ~~read~~ written are associated with.
<table>
<tr><th>Bits</th><th>Attribute</th><th>Definition</th></tr>
<tr><td>07</td><td>Incompressible</td><td>If set to ‘1’, then data is not compressible for the logical blocks indicated. If cleared to ‘0’, then no information on compression is provided.</td></tr>
<tr><td>06</td><td>Sequential Request</td><td>If set to ‘1’, then this command is part of a sequential write that includes multiple Write commands. If cleared to ‘0’, then no information on sequential access is provided.</td></tr>
<tr><td>05:04</td><td>Access Latency</td><td>
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00b</td><td>None. No latency information provided.</td></tr>
<tr><td>01b</td><td>Idle. Longer latency acceptable.</td></tr>
<tr><td>10b</td><td>Normal. Typical latency.</td></tr>
<tr><td>11b</td><td>Low. Smallest possible latency.</td></tr>
</table>
</td></tr>
<tr><td>03:00</td><td>Access Frequency</td><td>
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>0000b</td><td>No frequency information provided.</td></tr>
<tr><td>0001b</td><td>Typical number of reads and writes expected for this LBA range.</td></tr>
<tr><td>0010b</td><td>Infrequent writes and infrequent reads to the LBA range indicated.</td></tr>
<tr><td>0011b</td><td>Infrequent writes and frequent reads to the LBA range indicated.</td></tr>
<tr><td>0100b</td><td>Frequent writes and infrequent reads to the LBA range indicated.</td></tr>
<tr><td>0101b</td><td>Frequent writes and frequent reads to the LBA range indicated.</td></tr>
<tr><td>0110b</td><td>One time write. E.g. command is due to virus scan, backup, file copy, or archive.</td></tr>
<tr><td>0111b – 1111b</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

Modify a portion of Figure 92 (Identify Namespace) as shown below:

32	O	**Format Progress Indicator (FPI):** If a format operation is in progress, this field indicates the percentage of the namespace that remains to be formatted. Bit 7 if set to '1' indicates that the namespace supports the Format Progress Indicator defined by bits 6:0 in this field. If this bit is cleared to '0', then the namespace does not support the Format Progress Indicator and bits 6:0 in this field shall be cleared to 0h. Bits 6:0 indicate the percentage of the Format NVM command ~~namespace~~ that remains to be completed ~~formatted~~ (e.g., a value of 25 indicates that 75% of the Format NVM command ~~namespace~~ has been completed ~~formatted~~ and 25% remains to be completed ~~formatted~~). A value of 0 indicates that the namespace is formatted with the format specified by the FLBAS and DPS fields in this data structure and there is no Format NVM command in progress.

Modify Figure 31 as shown below:

Figure 1: Status Code – Generic Command Status Values, NVM Command Set

Value	Description
80h	**LBA Out of Range:** The command references an LBA that exceeds the size of the namespace.
81h	**Capacity Exceeded:** Execution of the command has caused the capacity of the namespace to be exceeded. This error occurs when the Namespace Utilization exceeds the Namespace Capacity, as reported in **Error! Reference source not found.**.
82h	**Namespace Not Ready:** The namespace is not ready to be accessed. The Do Not Retry bit indicates whether re-issuing the command at a later time may succeed.
83h	**Reservation Conflict:** The command was aborted due to a conflict with a reservation held on the accessed namespace. Refer to section **Error! Reference source not found.**.
84h	**Format In Progress:** ~~The namespace is currently being formatted.~~ A Format NVM command is in progress on the namespace. The Do Not Retry bit shall be cleared to '0' to indicate that the command may succeed if it is resubmitted.
85h – BFh	Reserved

Modify Figure 89 as shown below:

Figure 89: Identify – Command Dword 10

Bit	Description
31:16	**Controller Identifier (CNTID):** This field specifies the controller identifier used as part of some Identify operations. If the field is not used as part of the Identify operation, then host software shall clear this field to 0h for backwards compatibility (0h is a valid controller identifier). < ADD BLANK LINE > Controllers that support Namespace Management shall support this field. This field is used for Identify operations with a CNS value of 12h or 13h. This field should be cleared to 0h for Identify operations with a CNS value of 00h, 01h, 02h, 10h, and 11h.
15:08	Reserved
07:00	**Controller or Namespace Structure (CNS):** This field specifies the information to be returned to the host. Refer to **Error! Reference source not found.**.